# An Artificial Intelligence Approach to VLSI Design

Thaddeus J. Kowalski

Kluwer Academic Publishers

[illegible]
TK
7874
.K675
1985
c. 2

**Distributors for North America:**
Kluwer Academic Publishers
101 Philip Drive
Assinippi Park
Norwell, MA 02066, USA

**Distributors outside North America:**
Kluwer Academic Publishers Group
Distribution Centre
P.O. Box 322
3300 AH Dordrecht
THE NETHERLANDS

**Library of Congress Cataloging in Publication Data**

Kowalski, Thaddeus J.
An artificial intelligence approach to VLSI design.

(The Kluwer international series in engineering and computer science; SECS 4)
Bibliography: p.
Includes index.
1. Integrated circuits — Very large scale integration.
2. Artificial intelligence. 3. Expert systems.
4. Digital electronics. I. Title. II. Series.
TK7874.K675 1985 621.3819'5835 85-7581
ISBN 0-89838-169-X

2-4

Fourth printing, 1988

Printed in the United States of America

# Chapter 2

# KNOWLEDGE-BASED EXPERT SYSTEMS

During the past decade KBESs have been developed by researchers in artificial intelligence to help solve problems whose structures do not lend themselves to recipe-like solutions. These systems differ from previous efforts in problem solving by effectively coping with the enormous search spaces of alternatives found in real-world problems. The key to success in KBESs is the ability to use domain-rich knowledge to recognize familiar patterns in the current problem state and act appropriately. This tool is based on the premise that humans solve problems by recognizing one of many familiar patterns in the current situation and applying the appropriate actions when this pattern occurs. Their recognition of the pattern is not based on the complete current situation, nor is it recognized with absolute certainty. The presence and absence of patterns can be used to help establish and rule out actions for a given situation.

Researchers have developed many KBESs whose features[35] have matured and grown into usable engineering tools.[36, 37, 38] These systems exploit special knowledge to solve difficult problems in many domains. Examples are: DENDRAL, mass-spectrum analysis; MYCIN, medical diagnosis; PROSPECTOR, mineral exploration; R1, VAX computer configuration; to name only a few. This chapter introduces general components of KBESs, describes their general features, describes two implementations, and provides references to several other systems. Chapter 3 discusses how the KBES approach has been used to develop an intelligent CAD tool, DAA.

## 2.1 General Components of a Knowledge-Based Expert System

The problem domains and features of the existing KBESs differ widely, but many have three components in common: a working memory, a rule memory, and a rule interpreter.

*2.1.1 The working-memory component.* The working memory is a collection of attribute-value pairs that describe the current situation. They resemble the data structures in conventional programming languages:

```
struct  foo {
        <attribute 1> = <value 1>;
        ...
        <attribute n> = <value n>;
};
```

Some systems also represent goals and sub-goals as named attribute-value pairs in working memory.

*2.1.2 The rule memory component.* The rule memory is a collection of conditional statements that operate on elements stored in the working memory. The statements resemble the conditional statements of conventional programming languages:

```
IF:
        <antecedent 1>
        ...
        <antecedent n>

THEN:
        <consequence 1>
        ...
        <consequence m>
```

The rule memory is a collection of knowledge *chunks* about a particular problem domain. Most rule-based systems contain hundreds of rules that have been *painfully* extracted from months of interviewing experts. Acquiring knowledge from experts is difficult because although they are skillful at doing the task, they are not effective at explaining precisely how they do the task. To give a feel for the number of rules: MYCIN has about four-hundred fifty rules, R1 has about eight-hundred fifty rules (the new version XCON has three-thousand three-hundred rules),[39] and PROSPECTOR has about one-thousand six hundred rules. Systems like MYCIN have added debugged rules at an average rate of about two per week. This in no way accounts for the hundreds of rules that came and went in the debugging of the rule memory.

*2.1.3 The rule interpreter component.* The rule interpreter pattern matches the working-memory elements against the rule memory to decide what rules apply to the given situation. Some tool-kits allow a degree of certainty to be associated with each consequence that suggests the degree to which the consequence follows from the antecedents. Others allow a pair of certainties to be associated with each consequence that suggests both how sufficient the presence of the antecedents are for establishing the consequence and how sufficient the absence of the antecedents are for not establishing the consequence. Still others apply the consequences with absolute certainty if the antecedents are present in working memory. The selection process of rules can be data driven, goal driven, or a combination of data and goal driven.

A data-driven selection process looks through the rule memory for a rule whose antecedents match elements in the working memory. This is also called forward-chaining or antecedent reasoning. The consequences of the rule are applied, and the process is repeated until no more rules apply or until a rule explicitly stops the process.

A goal-driven selection process looks through the rule memory for a rule whose consequences can achieve the current goal. This is also called backward-chaining or consequent reasoning. If the antecedents of this rule match the elements in working memory, then the consequences are applied and the goal is satisfied. If the antecedents of this rule are not all present in working memory, then the missing antecedent replaces the current goal and the process is repeated until either all the sub-goals are satisfied or no more rules are applicable. If no more rules are applicable, the user can be queried for missing information to place in working memory. This backward-chaining reason also facilitates explanations of how the system had reached a particular conclusion and why it needed certain pieces of knowledge.

### 2.2 General Features of a Knowledge-Based Expert System

Separating expert knowledge from the reasoning mechanism implies several general features common to knowledge-based expert systems. The knowledge engineer can incrementally add new rules and refine old ones because the rules are chunks of domain knowledge and have minimal interaction with other rules in the rule memory. The rule interpreter can be queried about *why* it needs an additional piece of information and *how* it solved a problem by describing the goals and the rules it has used to solve a problem. A rule interpreter can be developed for aiding the acquisition of knowledge,[40] by checking its own rule memory for oddity, consistency and omitted areas. A rule interpreter can also be developed for aiding the instruction of knowledge,[41] by trying to determine what knowledge students seem to possess and how that knowledge corresponds to its rule memory. Finally, the same rule interpreter may be used by many rule memories and working memories to develop KBESs for different problem domains.

### 2.3 Example Knowledge-Based Expert Systems

During the past decade many KBESs have been developed for such purposes as interpretation, diagnosis, monitoring, prediction, planning and design. This section discusses two examples — a medical diagnostic system, MYCIN, and a system used to configure VAX computers, R1, and provides references to further readings in Tables 1 and 2. The commentary provides a description of the task and a discussion of the rule interpreter with its current states.

**Table 1.** A DECADE OF SYSTEMS

| System | Domain |
|---|---|
| INTERNIST[42] | diagnosis in medicine |
| MYCIN[43] | diagnosis in medicine |
| PUFF[37] | diagnosis in medicine |
| GUIDON[44] | computer-aided instruction in medicine |
| VM[45] | measurement interpretation in medicine |
| SACON[46] | diagnosis in structural engineering |
| PROSPECTOR[47] | diagnosis in geology |
| DENDRAL[48] | mass-spectrum analysis in chemistry |
| SECHS[49] | organic chemistry |
| SYNCHEM[50] | chemistry |
| SADD[51] | electronics |
| EL[52] | circuit analysis |
| PALLADIO[34] | VLSI design |
| SOPHIE[53] | computer aided instruction in electronics |
| MOLGEN[54] | problem solving and planning in genetics |
| NEWTON[55] | problem solving and planning in mechanics |
| PECOS[56] | problem solving and planning in programming |
| DART[57] | diagnosis in computer faults |
| R1[58] | configuring VAX computers |
| XSEL[59] | computer sales person assistant |
| CONCHE[60] | knowledge acquisition |
| TEIRESIAS[40] | knowledge acquisition for MYCIN systems |
| HEARSAY-II[61] | speech recognition |

**Table 2.** A DECADE OF TOOLS

| System | Domain |
|---|---|
| AGE[62] | blackboard model tool-kit |
| EMYCIN[63] | MYCIN tool-kit |
| EXPERT[64] | diagnosis in medicine |
| HEARSAY-III[65] | HEARSAY-II tool-kit |
| KAS[66] | PROSPECTOR tool-kit |
| META-DENDRAL[67] | DENDRAL tool-kit |
| NEOMYCIN[68] | computer aided instruction in MYCIN |
| AMORD[69] | an EL tool-kit |
| OPS3[70] | an OPS family tool-kit |
| OPS5[28] | an OPS family tool-kit |
| ROSIE[71] | a RITA tool-kit |

*2.3.1 The MYCIN example.* MYCIN is an interactive program for medical diagnosis. It produces diagnoses of infectious diseases, particularly blood infections and meningitis infections, and advises the physician on antibiotic therapies for treating those infectious diseases. During the consultation, which is conducted in a stylized form of English, the physician is asked only for patient history and laboratory test results. The physician may ask for an explanation of the diagnosis and the reason a laboratory test result is required.

MYCIN uses a goal-driven rule interpreter to scan a rule memory of about five hundred rules covering meningitis and blood infections. Each rule has a certainty factor that suggests the strength of the association or degree of confidence between the antecedents of the rules and the consequences of the rules. MYCIN has equaled the performance of nationally recognized experts in diagnosing and recommending therapy for meningitis and blood infections. However, it is not clinically used because the human factors of its interface do not save the doctors any time and its knowledge is limited to these two domains.

*2.3.2 The R1 example.* R1 is a program for configuring VAX computers. It organizes the customer selected components by location relative to other components or to the rooms in which the system will be housed, and it specifies the cabling required to connect pairs of components. This configurer is given only the set of components selected by a customer and produces the list of missing components, components in each CPU cabinet, components in each UNIBUS box, distribution panels in each cabinet, the floor layout, and cabling requirements.

R1 uses a data-driven rule interpreter to scan a rule memory of about eight hundred rules covering component configuration. These rules are written in the OPS5 programming language and use a direct association between antecedents and consequences. It takes about seventy five seconds on a VAX 11/780 to configure a typical order of about 90 components. R1 has equaled the performance of Digital Equipment Corporation experts in configuring VAX computers and is currently being used in a production mode.

## 2.4 KBES Summary

During the past decade many KBESs have been developed by researchers in artificial intelligence that assist experts in solving real-world problems such as interpretation, diagnosis, monitoring, prediction, planning and design. These systems differ from previous efforts in problem solving by effectively coping with the enormous search spaces of alternatives found in real-world problems. The key to success in KBESs is the ability to use domain-rich knowledge to recognize familiar patterns in the current problem state and act appropriately. Development of KBESs is aided by the separation of working memory, rule memory, and rule interpreter, which enables *how* and *why* questions to be asked. Researchers have developed many KBESs whose features have matured and grown into usable engineering tools. Most importantly, this tool is appropriate to the design domain because there are human experts available whose knowledge has been gained through experience and who can teach this knowledge through apprenticeship. Furthermore, the knowledge required to do the task is extensive and requires the type of organization provided by the KBES approach.

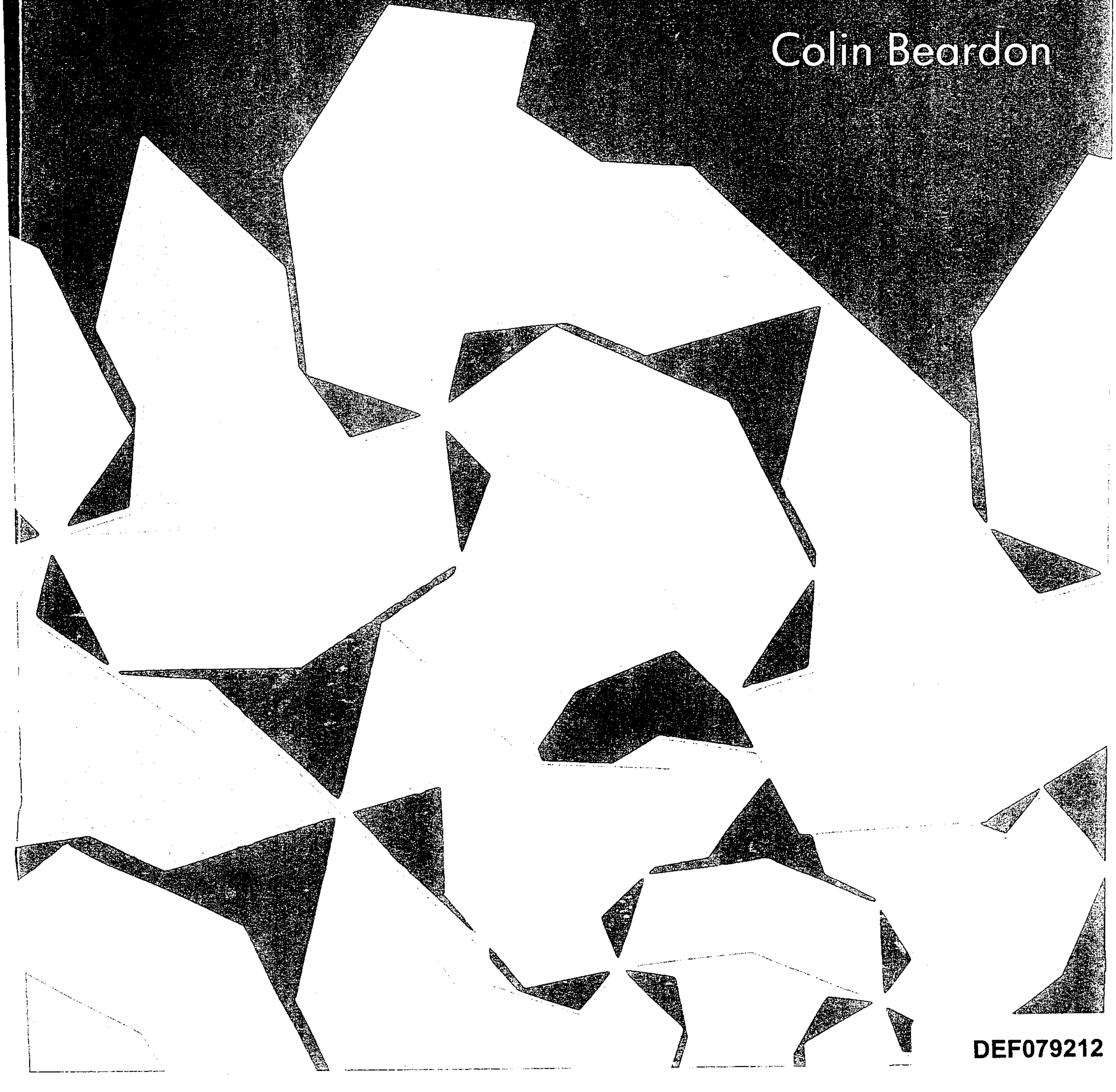
**26**

ELLIS HORWOOD SERIES IN ARTIFICIAL INTELLIGENCE FOUNDATIONS AND CONCEPTS

# ARTIFICIAL INTELLIGENCE TERMINOLOGY

## a reference guide

Colin Beardon



First published in 1989
**ELLIS HORWOOD LIMITED**
Market Cross House, Cooper Street,
Chichester, West Sussex, PO19 1EB, England
*The publisher's colophon is reproduced from James Gillison's drawing of the ancient Market Cross, Chichester.*

**Distributors:**
*Australia and New Zealand:*
JACARANDA WILEY LIMITED
GPO Box 859, Brisbane, Queensland 4001, Australia
*Canada:*
JOHN WILEY & SONS CANADA LIMITED
22 Worcester Road, Rexdale, Ontario, Canada
*Europe and Africa:*
JOHN WILEY & SONS LIMITED
Baffins Lane, Chichester, West Sussex, England
*North and South America and the rest of the world:*
Halsted Press: a division of
JOHN WILEY & SONS
605 Third Avenue, New York, NY 10158, USA
*South-East Asia*
JOHN WILEY & SONS (SEA) PTE LIMITED
37 Jalan Pemimpin # 05–04
Block B, Union Industrial Building, Singapore 2057
*Indian Subcontinent*
WILEY EASTERN LIMITED
4835/24 Ansari Road
Daryaganj, New Delhi 110002, India

**British Library Cataloguing in Publication Data**
Artificial intelligence terminology: a reference guide. —
(Ellis Horwood series in artificial intelligence
1. Artificial intelligence
I. Beardon, Colin
006.3

**Library of Congress CIP data available.**

ISBN 0–7458–0718–6 (Ellis Horwood Limited–Library Edn.)
ISBN 0–7458–0763–1 (Ellis Horwood Limited–Student Edn.)
ISBN 0–470–21601–8 (Halsted Press)

Printed in Great Britain by Hartnolls, Bodmin

## AI

[ * ] AI is an abbreviation of '*Artificial Intelligence*'.

## AKO-link

**[knowledge representation]** Generally, an AKO-link ('A-Kind-Of') is another name for an *inst-link* within a *semantic network*. In *KRL*, however, 'AKO-links' cover both *ISA-links* and inst-links.

**See also:** inheritance hierarchy, inst-link, ISA-link, semantic network.

## algorithm

**[software engineering]** An algorithm is a specification, written in some notation, for a series of steps which, if carried out, will result in a particular task being performed. It expresses a general method for achieving a particular result, some examples of everyday algorithms being knitting patterns and recipes. Algorithms are used in computing to break down a given problem into a number of steps which can then be written as programming language statements.

**See also:** program, stepwise refinement, structured programming, top-down design.

## allophone

**[speech processing]** Each *phoneme* can be realised in actual speech in a variety of ways depending, for example, on the sounds that precede or follow it. The set of alternative realisations of a phoneme is called an 'allophone'.

**See also:** phoneme, segmentation.

## alpha-beta pruning

**[game playing]** Alpha-beta pruning is a technique applied to improve the efficiency of systems that play two-person *zero-sum* games. It is applicable where a *depth-first search* strategy with limited depth is employed using *evaluation functions*. It is based on the principle that, having fully evaluated one move, other moves are not worth exploring further once a counter-move has been discovered that makes them less attractive than some fully evaluated move. The name 'alpha-beta' acknowledges the fact that a move that is attractive to one player (alpha) is proportionately unattractive to their opponent (beta). The technique requires no extra knowledge to be applied yet can significantly improve efficiency, especially if the most likely move is explored first.

**See also:** game playing, game tree, minimax.

## antecedent

**[linguistics]** The antecedent of an anaphoric expression is the earlier linguistic expression which determines its *reference*.

**See also:** anaphor, anaphora.

**[logic]** In a conditional proposition, standardly expressed in the form

$$if\ P\ then\ Q$$

or in *propositional calculus*

$$P \rightarrow Q$$

the antecedent is the component proposition ($P$) immediately following the 'if', or, in logical notation, preceding the '→'.

**See also:** consequent.

**[problem solving]** The term 'antecedent' can be used to refer to a small procedure that employs *forward chaining* techniques within a predominantly *backward chaining* system.

**See also:** antecedent rule, antecedent theorem.

## antecedent rule

**[expert systems]** An antecedent rule is a rule that performs *forward chaining* within a system that generally employs *backward chaining*. Antecedent rules are usually included for efficiency, to allow certain facts to be explicitly asserted, rather than being repeatedly inferred by the application of rules. Antecedent rules are employed in *MYCIN*.

**See also:** assert, backward chaining, forward chaining, inference.

## antecedent theorem

**[PLANNER]** An antecedent theorem in *PLANNER* behaves like an *if-added demon*. It allows the programmer to specify actions that are to be performed whenever a fact of a particular kind is *assert*ed. It can be used to explicitly record certain properties of objects by asserting appropriate facts, thus avoiding excessive use of inferencing where a property is often referred to. For example, an antecedent theorem might assert

**animate(X)**

whenever a fact of the form

**male(X)**

was asserted.

**See also:** assert, if-added demon, inheritance hierarchy, PLANNER.

DEF079236

## connectivity

**[connection science]** Connectivity refers to the pattern of *weighted* links between units in a connectionist network. There may be partial connectivity, where some of the units are connected to one another, or there may be complete connectivity, where every unit is connected to every other unit.

## CONNIVER

**[programming languages & problem solving]** CONNIVER is a *problem solving* language that was designed to overcome the automatic *backtracking* in *PLANNER* by enabling the programmer to direct the flow of control in the *program*.

**See also:** backtracking, PLANNER.

**Ref:** Sussman,G.J. & McDermott,D. (1972) *Why Conniving is better than Planning.* A.I.Memo 255A, MIT, Camb, Mass

## connotation

**[philosophy]** J.S.Mill distinguished between the connotation and the *denotation* of a word. In using a word, such as chair, one implies that the things to which it applies have certain properties, such as 'being a piece of furniture' and 'designed for sitting on'. Those properties make up the connotation of the word.

**See also:** denotation, intension, sense.

## CONS

**[LISP]** CONS is a *LISP function* that takes an *S-expression* and a *list* and creates a new list consisting of the S-expression as its *head*, followed by the old list as its *tail*, e.g.

$$cons('a\ \ '(b\ \ c))$$

will produce the list

$$(a\ \ b\ \ c)$$

**See also:** CAR, CDR, dotted pair, LISP, list, quote.

## consequent

**[logic]** In a conditional proposition, standardly expressed in the form

$$if\ P\ then\ Q$$

or in *propositional calculus*

$$P \rightarrow Q$$

the consequent is the component proposition ($Q$) immediately following 'then', or, in logical notation, following '$\rightarrow$'.

**See also:** antecedent.

## expectation based parser

**[computational linguistics]** At various stages when processing text, a top-down *parser* will be faced with a number of tests that can be applied. An expectation-based parser uses what it has already parsed to restrict its choice at this point by only carrying out tests that it has predicted. The entry for each word in the *lexicon* will contain the categories to which it belongs and some expectations about what may appear next and actions to be performed if the expectations are met. The parser is initialised to a *set* of expectations at the start of the sentence before processing commences.

For example, the word 'the' might expect a noun to follow and, if it does, it can record that a noun-phrase has been found. The process of setting up expectations based on the lexical entries for words, seeing if these expectations are satisfied, and carrying out their actions, is repeated until the end of the sentence is predicted and found, or the input does not match any expectation.

**See also:** categorial grammar, English Language Interpreter, word-expert parser.

## EXPEL

**[conceptual dependency]** EXPEL is a *primitive act* within *conceptual dependency* representation that represents the expulsion of an object from the body of an animal.

**See also:** conceptual dependency, primitive act.

## EXPERT

**[knowledge representation]** EXPERT is a simple *knowledge representation* language designed for general use in representing expert knowledge. It is best suited to diagnosis or classification problems, which is one of the reasons it is most widely used in medical applications. EXPERT is implemented in FORTRAN which makes it portable and efficient.

**See also:** KRL.

**Ref:** Weiss, S. M., Kulikowski, C. A. (1979) EXPERT: a system for developing consultation models. In: *Proceedings of the sixth IJCAI*, Tokyo

## expert system

**[ * ]** An expert system is a computer *program* which attempts to embody the knowledge and decision-making facilities of a human expert in order to carry out a task generally regarded as requiring some degree of human expertise. Expert systems usually operate within restricted domains, some well known examples being medical diagnosis and mineral exploration. Their reasoning is essentially *heuristic*, in that an *algorithmic* approach which is guaranteed to produce an *optimal* solution to a problem, is not feasible. An expert system will generally consist of a *rule base*, an *inference* engine and a user interface (which will generally provide an explanation facility).

**See also:** CADUCEUS, CENTAUR, DENDRAL, INTERNIST, knowledge-based system, MACSYMA, MYCIN, PROSPECTOR, rule base, shell, TAXMAN, XCON.

## explanation-based learning

**[machine learning]** Explanation-based learning (EBL) involves the learning of a concept from a single example. The *domain* theory held by the system is used to explain why the example is an instance of the concept under study. In explanation-based generalisation, a subset of EBL, this explanation is generalised so that the resulting rule may be applied to similar, though not necessarily identical, situations.

**See also:** concept learning, learning by induction, similarity-based learning.

## exponential

**[mathematics]** The term 'exponential' in mathematics indicates that a term is raised to a power (normally greater than 1). For example, '$2^3$' is expressed exponentially, whereas '8' is not.

**[software engineering]** The term exponential is used to describe algorithmic *complexity*. An *algorithm* whose complexity for **n** inputs is measured in terms of $C^n$, for some constant **C**, is called an exponential algorithm. Most exponential algorithms are not feasible for any but the smallest input data problems.

**See also:** algorithm, combinatorial explosion, complexity, order of complexity, polynomial.

## extended Horn clause

**[logic]** For the purposes of practical *logic programming*, *Horn clauses* are over-restrictive and some extensions to Horn clause notation have been adopted by logic programming languages such as *PROLOG*. For example, a Horn clause, by definition, allows at most one unnegated *literal*. When a Horn clause is converted to PROLOG notation this means that there can be no negated terms in the *body* of the PROLOG *clause*. To get around this limitation a special form of *negation* (called *negation-as-failure*) is introduced that can be applied to terms in the body of a clause. This is one way in which PROLOG notation is an extension of Horn clause notation, others include the use of operators such as *cut*.

**See also:** clause, cut, Horn clause, negation-as-failure.

## inductive logic

[logic] In an inductive logic the conclusion follows from the premises only with a degree of probability and an additional premise may change the probability of the conclusion. The *conjunction* of the premises and the *negation* of the conclusion is not a contradiction.

Thus typically one may infer from a finite number of statements such as "$raven_1$ is black", "$raven_2$ is black" ... "$raven_n$ is black", the conclusion "all ravens are black". Clearly the conclusion only follows with a certain probability and the addition of the new premise "$raven_{n+1}$ is not black" would invalidate the *inference*. Inductive logic is contrasted with *deductive logic*.

**See also:** BACON, concept learning, deductive logic, learning by induction, non-monotonic reasoning.

## inference

[logic] An inference is the drawing of a conclusion, or alternatively is just the conclusion drawn. A logical inference can be expressed as a relationship among a set of sentences or propositions.

**See also:** argument, deductive logic, inductive logic.

## inference engine

[expert systems] An inference engine is that part of an expert system that interprets rules and facts in order to make *inferences*. Depending upon the system, it may be driven by a specific question (*backward chaining*) or attempt to make inferences from a set of *data* (*forward chaining*). Often a combination of both methods is employed.

**See also:** chaining, expert system, inference, rule base.

## infinite loop

[software engineering] An infinite loop is a series of programming language statements which will repeat indefinitely, usually due to an error in the *program* code. The failure to alter a control *variable* within an iterative statement is a typical cause of infinite loops. For example, if $x = 1$, then the loop,

$$\textit{while} \quad x <= 10 \quad \textit{do} \quad \textit{write}\ ('\textit{hello}')$$

will repeat indefinitely.

**See also:** tractable.

## kludge

[ * ] A kludge is a part of a mechanism whose behaviour cannot be simply explained by reference to the principles adopted by the rest of the mechanism, or by universal principles. Kludges can be best explained by a number of *domain-specific* rules. Insofar as the mechanism operates within a *paradigm*, the term 'kludge' can refer to aspects of its behaviour that are better explained by theories external to the paradigm. For example, aspects of human behaviour may be better explained in terms of human evolution, rather than as integral aspects of a *symbolic information processing* system.

## knowledge acquisition

[expert systems] Knowledge acquisition is the activity of attempting to make explicit the relevant knowledge of a human expert. Often the objective is to encode the expertise of the human expert in terms of the formal notation of an expert system (e.g. rules, *metarules* and facts). This can be broken down into three stages: eliciting knowledge, encoding it in some *canonical representation*, and validating it by making sample *inferences* which can be checked by domain experts and give rise to amendments.

See also: expert system, knowledge elicitation, knowledge engineer, TEIRESIAS, validation.

## knowledge based system

[ * ] A knowledge based system (KBS) can be contrasted to a *database* or *information retrieval* system. In the latter cases information is retrieved from a store in response to precise requests and only the information described is presented. In KBS a more general goal can be stated and the system may make any relevant *inferences* in order to provide an appropriate answer. The result may not carry absolute certainty, but should represent an intelligent response to a problem given the knowledge available to the system.

The term 'Knowledge based systems' is sometimes used synonymously with '*expert systems*', though it is usually taken to be a more general term incorporating, for instance, some *natural language processing* systems.

See also: database system, expert system, information retrieval.

## knowledge elicitation

[expert systems] Knowledge elicitation is the task of trying to make explicit the knowledge behind the performance of experts in their particular *domain*. One method is to ask the experts to explicitly state the rules they work by, but this does not always work well as their expertise is essentially performative, rather than *declarative*. An alternative approach is for the expert to give examples of their reasoning and for rules to be derived from these. In other cases, rules may be found ready formalised in text books or similar literature.

See also: expert system, knowledge acquisition, knowledge engineer.

## production system

[expert system] A production system is an *inference* system that consists of three parts: the working memory, which represents the *current state* of the inference (also known as the 'context' or 'short-term memory buffer'); the production memory, which represents the inference rules; and the rule *interpreter*, which applies the rules to the working memory. At each cycle all of the rules in the production memory are examined to see if their conditions are satisfied in the working memory and a list of rules that can be fired is produced. A *conflict resolution* strategy is then employed to decide which rule to fire. The action part of the chosen rule is executed and the working memory is updated as a result. This process continues until either the working memory contains the *goal*, or no more rules can be fired.

See also: DENDRAL, MYCIN, OPS5, production rule, PROSPECTOR.

## program

[computer systems] A computer program consists of a series of definitions and/or instructions conforming to the *syntax* of a given programming language which, when executed on, or interpreted by, a computer will perform a certain task. A program is often defined as being a marriage between *data structures* and *algorithms*. Because programs will be executed on some machine, they are essentially *procedural*, yet insofar as they describe the problem in terms of its own *domain*, they may also aspire to being *declarative* or functional.

See also: algorithm, data structure, declarative, procedural.

## program synthesis

[problem solving & programming] Program synthesis (also known as *automatic programming*) involves the generation of computer program code given some description of the problem in another form. This description might be in the form of examples of behaviour, a statement in a *formal language* (such as *logic*), a description in natural language, or result from an interactive dialogue.

## PROGRAMMAR

[computational linguistics & programming languages] PROGRAMMAR is a *natural language parsing* system developed by T.Winograd for the *SHRDLU* project. Though formally similar to an *ATN*, PROGRAMMAR is based upon ideas from *systemic grammar*. The language to be parsed is defined *procedurally*, enabling special purpose mechanisms to be introduced if required. It attempts to be as deterministic as possible, though facilities do exist for *backtracking* and trying alternative paths.

See also: augmented transition network, deterministic parsing, SHRDLU, systemic grammar.

## ROSIE

[expert systems] ROSIE is an artificial, English-like language for expressing facts and *inference* rules within expert systems. Though some of its constructs are quite formal, it can be read and interpreted *declaratively* once certain conventions are learned.

## rote learning

[learning] Rote learning refers to knowledge accumulation involving no transformations or processing of the information received. Forms of rote learning include learning by being programmed and learning by memorisation.

**See also:** learning from instruction.

## rule base

[expert systems] Just as a *database* is essentially a collection of data, so a rule-base is a collection of rules. Typically the term is used to refer to the set of rules in an *expert system*.

**See also:** production memory.

## rule-based system

[expert systems] A rule-based system is any system that represents knowledge as a set of *rules* that can be intepreted in a uniform way by applying them to known *facts* to infer new facts. The objective of a rule-based system is to encode practical human knowledge in such a way that it can mimic the *problem-solving* abilities of a human expert.

**See also:** expert system, inference.

## runtime error

[programming] A runtime error is one which occurs after *compilation*, during the execution of the translated source program. This sort of error is often due to a *semantic* error or a memory violation.

**See also:** bug, compilation error, debugging, semantic.

## runtime stack

[programming languages] A runtime stack is a mechanism for giving a *procedural* interpretation of the way that one process can call another during the execution of a *program*. Any language that allows the recursive calling of processes (e.g. *procedures* in Pascal, *clauses* in *PROLOG*, *functions* in *LISP*) will need to keep track of suspended processes and their return points, and a *stack* is an obvious choice for this.

**See also:** clause, function, nondeterminism, procedure, recursion, stack.

## word recognition

**[cognitive psychology]** The study of word recognition is the study of the mental processes involved in identifying the meanings of words. Word recognition involves encoding the initial percept of the word, locating the appropriate entry in the *mental lexicon*, and selecting the appropriate meaning for a given context. Several alternative models of word recognition have been proposed.

**See also:** lexical access, logogen model, mental lexicon.

## working memory

**[cognitive psychology]** Working memory refers to that part of human memory that is sometimes called *short term memory*. Working memory is a transient store of information that can be maintained only through rehearsal. Use of the term 'working memory' emphasises the fact that the information is being held for use by mental procedures. Information can be retrieved from *long term memory* and held active in working memory as various computations are completed.

**See also:** long term memory, short term memory.

## workstation

**[computer systems]** Workstations are usually stand-alone, single-user computers, but thier power is such that they can normally support many users. Workstations sometimes lend themselves to specific applications, such as graphics and large scale numerical computation.

## WPE

**[programming languages]** WPE is an abbreviation of 'Warren Prolog Engine' which is the same thing as the *Warren Abstract Machine*.

27

MICROSOFT PRESS®

# COMPUTER DICTIONARY

THE COMPREHENSIVE STANDARD FOR BUSINESS, SCHOOL, LIBRARY, AND HOME

Microsoft PRESS

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Cataloging-in-Publication Data
Microsoft Press computer dictionary : the comprehensive standard for
business, school, library, and home.
p. cm.
ISBN 1-55615-231-0
1. Computers--Dictionaries. 2. Microcomputers--Dictionaries.
I. Microsoft Press.
QA76.15.M54 1991
004.16'03--dc20 91-9904
CIP

Printed and bound in the United States of America.

2 3 4 5 6 7 8 9 MLML 6 5 4 3 2 1

Distributed to the book trade in Canada by Macmillan of Canada, a division
of Canada Publishing Corporation.

Distributed to the book trade outside the United States and Canada by
Penguin Books Ltd.

Penguin Books Ltd., Harmondsworth, Middlesex, England
Penguin Books Australia Ltd., Ringwood, Victoria, Australia
Penguin Books N.Z. Ltd., 182–190 Wairau Road, Auckland 10, New Zealand

British Cataloging-in-Publication Data available.

**Acquisitions Editor:** Marjorie Schlaikjer
**Project Editor:** Mary Ann Jones
**Technical Editors:** David Rygmyr, Jeff Hinsch, Mary DeJong, Dail Magee, Jr.
**Manuscript Editor:** Pamela Beason
**Copy Editor:** Alice Copp Smith

DEF083324

lated on 8086/8088 or 80286 machines as well, but the performance is poor. *Compare* conventional memory, extended memory.

**expansion** A means of increasing the capabilities of a microcomputer by adding hardware designed to perform a task that is not built into the basic system. *Expansion* is generally used in reference to the addition of printed circuit boards (expansion boards) that plug into openings (expansion slots) inside the body of the computer. In IBM personal computers and others designed as open-ended systems, these slots enable expansion boards and associated devices to connect with and transfer information on the computer's main data highway, the bus. Computers with expansion slots can be equipped with as many additional pieces of hardware as there are slots available. *See also* expansion slot, open architecture.

**expansion board** A circuit board holding chips and other electronic components connected by conductive paths that is plugged into a computer's bus (main data-transfer path) to add functions or resources to the computer. Typical expansion boards add memory, disk-drive controllers, video support, parallel and serial ports, and internal modems. The simple terms *board* and *card* are used interchangeably by most people to refer to all expansion boards. *See also* expansion slot.

**expansion bus** *See* AT bus.

**expansion slot** A socket inside a computer console, designed to hold expansion boards and connect them to the system bus (data pathway). Most personal computers have from three to eight expansion slots, with the notable exceptions of the Apple Macintosh and Macintosh Plus, which have none, and the Macintosh SE, which has one. Expansion slots provide a means of adding new or enhanced features or more memory to the system. *See also* expansion board.

**expert system** A type of application program that makes decisions or solves problems in a particular field, such as finance or medicine, by using knowledge and analytical rules defined by experts in the field. Human experts solve problems by using a combination of factual knowledge and reasoning ability. In an expert system, these two essentials are contained in two separate but related components, a knowledge base and an inference engine. The knowledge base provides specific facts and rules about the subject, and the inference engine provides the reasoning ability that enables the expert system to form conclusions. Expert systems also provide additional tools in the form of user interfaces and explanation facilities. User interfaces, as with any application, enable people to form queries, provide information, and otherwise interact with the system. Explanation facilities, an intriguing part of expert systems, enable the systems to explain or justify their conclusions, and they also enable developers to check on the operation of the systems themselves. Expert systems originated in the 1960s; fields in which they are used include chemistry, geology, medicine, banking and investments, and insurance. *See also* artificial intelligence, intelligent database.

**exploded view** A type of display that shows a structure with its parts separated but drawn in relation to one another. See the illustration. A charting program, for example, could create an exploded pie chart by separating one segment from the rest of the pie. Similarly, an engineering or architectural drafting program could display a structure with its parts exploded outward so that the designer can view the pieces separately but as parts of the whole.

**exponent** In mathematics, a number that shows how many times another number is used as a factor in a calculation. Although exponents are commonly thought of as representing higher powers of a number, only positive exponents, as in $2^3$, indicate multiplication (2 times 2 times 2). Negative exponents, as in $2^{-3}$, indicate division (1 divided by $2^3$), and fractional exponents, as in $8^{1/3}$, indicate the root of the number (here the cube root of 8).

In the floating-point, or exponential, notation commonly used with computers, the exponent is a number that indicates the power of 10 to be multiplied by the number in the fixed-point portion of the notation. Essentially, the exponent in a floating-point number shows the number of places the decimal point is moved to the right (positive exponent) or to the left (negative exponent) when the numbe

28

# 1985 INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS

## PROCEEDINGS

Volume 2 of 3

SPONSORED BY

THE IECE (JAPAN) TECHNICAL GROUP ON CIRCUITS AND SYSTEMS

THE IEEE CIRCUITS AND SYSTEMS SOCIETY

JUNE 5–7, 1985
KYOTO HOTEL
KYOTO, JAPAN

IEEE Catalog No.
85CH2114 -7

Volume 1 Contains Pages 1-390
Volume 2 Contains Pages 391-1026
Volume 3 Contains Pages 1027-170

Proceedings of ISCAS 85

# The VLSI Design Automation Assistant: A Birth In Industry

*T. J. Kowalski* *D. J. Geiger* *W. H. Wolf* *W. Fichtner*

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

## ABSTRACT

This paper outlines our approach to automatic design of VLSI circuits from algorithmic descriptions. We discuss how to break the integrated-circuit design process into stages, and how to implement those stages as either knowledge-based expert systems or algorithmic programs. This paper briefly describes the elements of our approach and our plans for future work.

## INTRODUCTION

The complexity of integrated circuits has increased at an enormous rate, progressing from a few hundred components on a chip in the 1960's to hundreds of thousands in the mid-1980s, signaling the arrival of the VLSI era. This increase in complexity was made possible by advances that continue to shrink the dimensions of devices in integrated-circuit fabrication.

Although we can build complete systems on a single silicon chip, the design of a VLSI chip has become an expensive process. It can take several years to go from the conception of an architecture to the completion of the final product. This level of design complexity, created by the combinatorial explosion of details, makes it increasingly difficult to build cost-effective, low-volume, special-purpose VLSI systems.

In our opinion, the only solution to this problem is to build new CAD tools that are capable of automatically performing more of the synthesis process. In the last few years, we have been developing tools to act as expert assistants for the entire VLSI design process, from architecture definition to layout. We have aimed our approach at aiding the designer by automatically producing data paths and control sequences to implement the algorithmic system description within user-supplied constraints.

While our research is not complete, we have promising results at both the high and low end of the chip synthesis problem. At the high end of the design process, we have demonstrated that a knowledge-based expert system, KBES, can produce quality block-diagram designs from algorithmic descriptions of chips.[1] At the low end, we have built on work on symbolic layout[2,3,4] and on our work in compiling layout cells from simple descriptions.[5] This experience leads us to believe that a KBES floor planner is a good approach to bridge the gap between the high and low ends of chip design.

In this paper, we outline our approach to automatic design of VLSI silicon chips from algorithmic descriptions. After introducing our methodology for VLSI design synthesis, we discuss out approach in more detail. We conclude by presenting our future plans for the system.

## 1. METHODOLOGY

We look at hardware synthesis as the creation of a detailed representation from a more abstract representation. The designer originally specifies the chip's functionality as an algorithm independent of the style of implementation — bus, pipeline, or multiplexer — and target

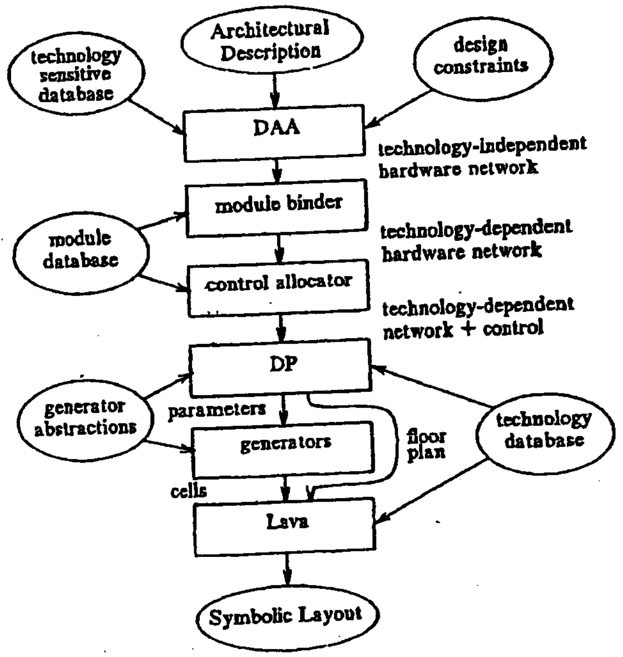

technology sensitive database
Architectural Description
design constraints
DAA
technology-independent hardware network
module binder
module database
technology-dependent hardware network
control allocator
technology-dependent network + control
DP
generator abstractions
parameters
generators
floor plan
technology database
cells
Lava
Symbolic Layout

Figure 1. THE VLSI DESIGN PROCESS

technology. The algorithm is transformed into a *technology-independent network*, built of modules of medium complexity — registers, adders, and multiplexers — that are not yet tied to a particular technology, along with a symbolic control description. This network is used to synthesize a *technology-dependent network* that includes a controller and instructions on how to build the technology-independent modules from components available in the target technology. The next step is the construction of a *floor plan*, which describes the placement and wiring of the chip in terms of the modules in the technology-dependent network. The floor plan guides the assembly of the final layout of the chip.

Our design process is outlined in more detail in Figure 1. Typically, the chip proceeds through six different tools as it is transformed from an algorithm to a layout:

- DAA — The design automation assistant, DAA,[6] is a KBES responsible for transforming the algorithmic description of the chip, written in ISPS,[7] into a technology-independent block diagram built of components such as registers, adders, multiplexers, buses, and a symbolic control specification for the implementation. The DAA also uses a technology-sensitive database and a set of user-supplied constraints. The technology-sensitive database contains expert knowledge about tradeoffs particular to the target technology, while the user-supplied constraints customize the design for the particular application. The DAA has been used to design several digital systems, including an IBM System/370. Work is underway to create pipeline style designs.
- Module binder — The module binder is an algorithmic program that implements the technology-independent modules from components available in the technology. The technology-dependent modules are either selected from the module database, or fabricated from equivalent simpler module database entries. The selection process is guided by user-supplied constraints on area, power, and speed information derived from the module database. The module binder is a current master's thesis at Carnegie-Mellon University by E. Dirkes.
- Control allocator — The control allocator[8] is an algorithmic program that adds the controller for the data path logic designed by the DAA and the module binder. It can target the controller towards either a microprogram engine or a PLA implementation. The controller description is represented either as a ROM or as a PLA module connected to the technology-independent controller component. The control allocator was developed at Carnegie-Mellon University, with work underway at AT&T Bell Laboratories to add pipeline style designs.
- DP — The data-path floor planner, DP, is a KBES responsible for the complete physical design of the chip. It designs a relative placement and interconnections for the technology-dependent description in a data path style, and determines the parameters required to generate the cell. The floor planning process is constrained by user-supplied restrictions on area and speed. Size and performance estimates for cells are derived from information in the technology and generator-abstraction databases. DP is in the early stages of development.
- Generators — A generator is an algorithmic program that produces a technology-specific custom cell layout for each set of cell-layout parameters. Generators can be as simple as programs that produce a PLA layout from a specification of the PLA's programming, or as complex as a maximum performance multiplier generator.[9] Useful knowledge about the generator's cells, such as area and delay, is stored in a generator abstraction. Generators are not implemented as KBESs, but are written by layout experts to encapsulate expert knowledge. Sample generators and programs exist to compile cells. Current work is focusing on the generator-abstraction database and expanding the library of generators.
- Lava — Lava is an algorithmic program that compacts a layout for each individual cell, then assembles the cells into a complete chip, stretching them as necessary to make the required connections. Lava generates a symbolic layout with transistors and other primitive components represented as symbols. All the components in the symbolic layout are placed at their final locations on the chip. The symbolic layout can be made into masks used to manufacture the integrated circuit. Lava, developed at Stanford University, has been used to design several chips.

## 2. DESIGN STRATEGIES

Breaking up the complete synthesis task into several programs obviously simplifies the implementation of these programs. Furthermore, it allows the choice of different computing environments for the various levels. While non-algorithmic programs such as the DAA and DP are best implemented in a KBES environment, algorithmic programs such as Lava and the generator software are best implemented in a programming-language environment. The segmentation also makes it easier for the designer to keep track of the chip's progress. The designer can examine the chip at each stage and, if necessary, try a different approach or constraint settings without carrying every alternative design to layout.

Our reasons for choosing generators over a general KBES layout system are twofold. First, we can write specialized generators corresponding to the hardware building blocks — ALUs, registers, etc. — and make use of expert knowledge about their physical design. Second, we believe that KBES layout generators cannot produce layouts for high-performance modules with the same success as has been achieved with algorithmic generators.[9]

In this section we outline the expert knowledge used by our programs and, when appropriate, algorithms used during synthesis.

### 2.1 DAA Knowledge

The DAA is a KBES that uses a knowledge base of over 500 rules and three estimating algorithms to synthesize an architectural implementation from an algorithmic description with constraints. The DAA starts with a data flow representation extracted from the algorithmic description. This representation resembles the internal description used by most optimizing compilers, and it is felt to be less sensitive when the same algorithm appears in a variety of writing styles.

The DAA produces a technology-independent hardware network description. This description is composed of modules, ports, links, and a symbolic microcode. The modules can be registers, operators, memories, and buses or multiplexers with input, output, and bidirectional ports. The ports are connected by links and are controlled by the symbolic microcode.

The DAA uses a set of temporally ordered subtasks to perform the synthesis task. It begins by allocating the base-variable storage elements — constants, architectural registers, and memories with their input, output, and address registers — to hardware modules and ports. Then a data flow BEGIN/END block is picked, and the synthesis operation assigns minimum delay information to develop a parallel design. Next, it maps all data flow operator outputs not bound to base-variable storage elements to register modules. Last, it maps each data flow operator with its inputs and outputs to modules, ports, and links. In doing so, the DAA avoids multiple assignments of hardware links; it supplies multiplexers where necessary. The last two mapping steps place the algorithmic description in a uniform notation for the expert analysis phase that follows.

The expert analysis subtask first removes registers from those data-flow outputs where the sources of the data-flow operator are stable. Operators are combined to create ALUs according to cost, partitioning, and pipeline estimators across the allocated design. The DAA also examines the possibility of sharing non-architectural registers. Where possible, it performs increment, decrement, and shift operations in existing registers. Where appropriate, it places registers, memories, and ALUs on buses. Throughout this subtask, constraint violations require tradeoffs between the number of modules and the partitioning of control steps. The process is repeated for the next data flow BEGIN/END block.

By using a KBES approach, the DAA uses a *weak method*[10] called *match* in place of extensive backtracking. The DAA uses match to explore the space of possible designs by extending a partial design from an initial state to a final state *without* any backtracking. The DAA proceeds through its major tasks in the same order for each problem; it never varies the order and it never backs up in any problem. This means that at any intermediate state, the DAA can determine how to extend the design to achieve an acceptable result.

## 2.2 Module Binding

The module binder chooses physically realizable technology-dependent cells to implement the technology-independent modules in the DAA's design. The cells are chosen from the module database on the basis of user-supplied constraints on functionality, power dissipation, delay, and area. The module binder may, for example, choose a ripple-carry adder when area is the major design constraint and a carry-look ahead adder when performance is the major design constraint. If the module database does not contain a cell with the right combination of functions, a combination of existing cells is used. For example, if the binder is unable to find a single module corresponding to some combination of an ADD and an OR, the binder will select separate ADD, OR and multiplexing cells to generate the required function. If the module database does not contain a cell of the proper bit width, an existing cell may be replicated. Finally, if the DAA has specified the use of buses, then the module binder will provide technology-dependent bus drivers.

## 2.3 Control Allocation

The controller of a digital system generates control signals while sequencing through a set of states. In our control allocator, a style-independent part generates the control signals, while a style-dependent part generates the sequencing information. This is beneficial because the style-independent part is the same regardless of the way the controller is implemented (PLA, microprogrammed controller, or random logic), while the style-dependent part changes with each different implementation style for the controller. The control allocator receives information from the technology-dependent hardware network and the module database to produce an output file for a specific style of controller. The structure and control specification file[11] provides the control signals and the technology-dependent data path, while the module database provides the required control line values. The current control allocator can produce PLA style output for the Berkeley[12] and the AT&T Bell Laboratories PLA generation tools.[13] It can also produce a microprogrammed style output for an AM2910 microengine.

## 2.4 Floor Planning

The floor planner finds the relative placement and a wiring plan for the bound modules that satisfy the design constraints. It uses this floor plan to generate the requirements and specifications that drive the cell generators.

The floor planner uses expert knowledge from two sources. One source is its knowledge base about chip assembly. This database includes rules for estimating performance and the area required for wiring. It also includes rules that guide the improvement of the floor plan — rules on changing the floor plan to improve area utilization or to reduce the delay through a timing path. The firing of these rules causes changes in the floor plan design and in the specifications for individual cells.

The second knowledge database describes the generators. Each generator is represented in the database by an abstraction that includes essential knowledge of the physical, electrical, and logical properties of the generated cells. This abstraction is necessary because the information is difficult to obtain directly from the generator. Much of the desired information — input load capacitance, aspect ratio, drive current, for example — is included only implicitly in the generator code and the floor planner has to make many decisions based on the information in the technology database. The desired cell could be generated and then measured, but this would be too time-consuming for chip synthesis. Knowledge about cell parameters is combined with physical constraints from the technology database to produce area and time estimates.

The floor planning process begins with the construction of an initial placement, guided primarily by the constraints of the physical interfaces between the cells. This initial placement is iteratively improved by application of the rules from the global knowledge base. Application of the rules produces new floor plans that are feasible to lay out and come successively closer to meeting the area and performance goals. During improvement, the floor planner also consults the generator abstractions to be sure that the postulated changes are feasible. Improvement steps modify the cell placement and the design of the cells. For instance, a rule may be applied to interchange two cells to shorten their buses, followed by another rule to reduce the cells' drive capability and reduce their area.

Once a floor plan that meets the goals has been devised, the Lava description of the floor plan is written and the generators are executed with parameters appropriate to produce the necessary cells.

## 2.5 Cell Generation and Assembly

In our work, cell generation and assembly is primarily an algorithmic process. The cells are generated by ic[14] programs and by the Dumbo cell compiler;[5] the cells are then assembled into the data path by the Lava compactor and assembler. The result is a symbolic layout with components at their final coordinates. The expert knowledge at this stage of design is embodied in the generator code, either the ic programs or the generic Dumbo cells.

Once the layout has been generated, it is important to extract parameters from it to test the validity of assumptions made during synthesis. If the assumptions, such as the predicted size of the cell, were incorrect, the design may not meet its goals. Extraction and testing can be done by layout analysis tools like GOALIE,[15] and simulators like MOTIS.[16]

## 3. FUTURE WORK

We are exploring the chip synthesis problem by producing data and control paths on working silicon from an algorithmic representation of a VLSI system. Using a hierarchical system of algorithmic and non-algorithmic techniques, our goal is to generate silicon from algorithms. This paper has reviewed the work in progress on several tools. The first tool, the DAA, is growing, but solidly in hand. Like human designers, the DAA becomes a better designer as its rule memory expands. Currently, DAA is being expanded to handle pipeline style designs. This work is in its early development stages, but shows great promise.

The module binder is nearing completion at Carnegie-Mellon University, while the control allocator is finished and has been used to design several PLA and microcoded controllers. The largest controller designed was for the RCA 1802 CMOS microprocessor. Changes will be needed in the control allocator if it is to design controllers for pipelined systems.

Our floor planning KBES is at an early stage. We are building a knowledge base about floor planning from interaction with chip designers and analysis of sample chip designs, such as the FLASH signal processor and other chips proprietary to AT&T Bell Laboratories. We are building the generator abstraction database from interviews with layout designers and analysis of existing generator programs. Soon, we will try to redesign our example chips with our synthesis tools and compare the automatic and hand designs.

After demonstrating working chip designs, we hope to experiment with high-level optimization based on layout analysis. Information about the physical design of the chip, gained during floor planning and cell generation, may be useful in optimizing the logic design or architecture of the chip. For example, if the floor planner is unable to produce a layout with an acceptable delay for a signal, the clocking scheme of the architecture may have to be redesigned. Automatic optimization is intriguing because the programs can generate and analyze much more information than can a human designer working alone, and because the new design can be reimplemented quickly. The problems are, however, daunting, and our goals in this area are long-range.

## 4. SUMMARY

This paper has described our approach to automated VLSI design. We have discussed how we break the integrated-circuit design process into stages, and how to implement those stages as programs. We have aimed for a methodology that automates as much of the design work as possible while still allowing the human designer to control the outcome of synthesis. In this way we can free the designer's creativity to solve the important problems in a chip design.

We strongly believe that only by automating more of the chip design process will we be able to exploit the possibilities of advanced VLSI fabrication technologies. We also believe that the combination of algorithms and knowledge-based expert systems approaches offers significant benefits when building computer-aided design tools for VLSI. Some synthesis problems are solved best by algorithms; others have not yielded good results in reasonable amounts of CPU time. KBESs use expert knowledge to solve these problems yielding good results in reasonable amounts of CPU time.

By combining expert systems with algorithmic approaches, we should be able to automate the entire custom chip design process. This development has profound implications. Automatically-produced designs can be guaranteed to be correct and can be produced far more quickly than if human intervention is required. The designer can either accept the automatic design or can use his knowledge to guide synthesis to a better result. Designers should be able to produce far better chip designs when aided by top-to-bottom synthesis aids than if they worked without CAD tools.

We still have much work to do, but we are excited by our results so far. We have demonstrated that the KBES methodology is an effective approach to high-level hardware synthesis. Our understanding of chip design leads us to believe that we can apply expert knowledge to the physical design of VLSI chips with similar success.

## 5. ACKNOWLEDGEMENT

The work described here has built on work done by us and others at Carnegie-Mellon University and Stanford University. At Carnegie-Mellon we would particularly like to thank S. Director, E. Dirkes, C. Forgy, D. Gatenby, C. Hitchcock, J. Lertola, J. McDermott, M. McFarland, J. Nestor, A. Newell, J. Rajan, D. Siewiorek, D. Thomas, and R. Walker for many ideas, probing questions and software support. At Stanford we owe particular thanks to P. Eichenberger, D. Perkins, and T. Saxe, who designed and implemented Lava, and to R. Mathews, J. Newkirk, R. Dutton, and J. Hennessy for valuable discussions and support.

We would like to thank J. Boddie, K. Chong, C. Davis, D. Ditzel, M. Maul, G. Mowery, A. Ross, C. Schneider, G. Williams, and A. Wilson for donating time to critique the various designs. We would also like to thank J. Hartung, D. Hill, D. McCreary, and H. Moscovitz for taking time to discuss the problems of chip synthesis.

## REFERENCES

[1] Kowalski, T. J. and Thomas, D. E., "The VLSI Design Automation Assistant: An IBM System/370 Design," *Design and Test of Computers* 1(1) pp. 60-69 (February, 1984).

[2] Mathews, R., Newkirk, J., and Eichenberger, P., "A Target Language for Silicon Compilers," *Compcon Proceedings*, pp. 349-353 (Spring, 1982).

[3] Weste, N., "Virtual Grid Symbolic Layout," *18th Design Automation Conference*, pp. 225-233 (1981).

[4] Wolf, W., *Two-Dimensional Compaction Strategies*, PhD thesis, Stanford University (March, 1984).

[5] Wolf, W., Newkirk, J., Mathews, R., and Dutton, R., "Dumbo, A Schematic-to-Layout Compiler," *Third Caltech Conference on VLSI*, pp. 379-394 (March, 1983).

[6] Kowalski, T. J., *The VLSI Design Automation Assistant: A Knowledge-Based Expert System*, PhD thesis, Department of Electrical and Computer Engineering, Carnegie-Mellon University (April 27, 1984).

[7] Barbacci, M. R., Barnes, G. E., Cattell, R. G., and Siewiorek, D. P., *The ISPS Computer Description Language*, Department of Computer Science, Carnegie-Mellon University (August 16, 1979).

[8] Geiger, D. J., *A Framework for the Automatic Design of Controllers*, Masters thesis, Carnegie-Mellon University (July, 1984). Research Report No. CMUCAD-84-34

[9] Chu, K and Ramautar Sharma, "A Technology Independent MOS Multiplier Generator," *Proceedings, 21st Design Automation Conference*, pp. 90-97 (1984).

[10] Newell, A., "Heuristic programming: ill-structured problems," pp. 360-414 in *Progress in Operations Research 3*, ed. Aronofsky, J., Wiley, New York (1969).

[11] Vasantharajan, J., *Design and Implementation of a VT-Based Multi-level Representation*, Masters thesis, Department of Electrical Engineering, Carnegie-Mellon University (February 10, 1982).

[12] De Micheli, G. and Sangiovanni-Vincentelli, A., "KISS: A Program For Optimal State Assignment of Finite State Machines," *International Conference on Computer-Aided Design*, pp. 209-211 IEEE, (November, 1984).

[13] Meyer, M. J. , Agrawal, P., and Phister, R. G., "A VLSI FSM Design System," *Proceedings of 21st Design Automation Conference*, pp. 434-440 IEEE, (June, 1984).

[14] Johnson, S. C. and Browning, S. A., *The LSI Design Language i*, 1980-1273-10.

[15] Syzmanski, T. G. and Van Wyck, C. J., "GOALIE: A Space-Efficient System for VLSI Artwork Analysis," *Proceedings, International Conference on Computer-Aided Design*, pp. 278-280 (1984).

[16] Chawla, B. R., Gummel, H. K., and Kozak, P., "MOTIS - An MOS Timing Simulator," *IEEE Transactions on Circuits and Systems* CAS-22(12) pp. 901-910 (Dec 1975).

29

# A RULE-BASED LOGIC CIRCUIT SYNTHESIS SYSTEM FOR CMOS GATE ARRAYS

Takao Saito, Hiroyuki Sugimoto, Masami Yamazaki(*), and Nobuaki Kawato

FUJITSU LABORARORIES LTD. (*) FUJITSU LTD.

1015, Kamikodanaka, Nakahara-ku, Kawasaki 211 Japan

## ABSTRACT

This paper presents a CMOS gate-array version of Digital System Design Language/ Synthesis eXpert (DDL/SX), a rule-based system for logic circuit synthesis. The system inputs technology-independent functional diagrams, and automatically generates conventional technology-dependent logic diagrams in order to eliminate time-consuming and error-prone tasks in logic design. Because the synthesis process was not clear enough to establish a fixed algorithm, a rule-based approach was adopted to develop the system. This approach made it easy to incrementally improve the system's capabilities by adding, modifying, or deleting design knowledge represented as rules. Experimental use of the system revealed that the automatically generated logic design is almost as good as a manual design, and the design time is reduced by a factor of four.

## 1. INTRODUCTION

Recent advances in semiconductor technology have led to a significant increase in the size and complexity of digital systems and it is more important to increase reliability and shorten design time. So it is necessary to extend the capability of CAD systems to assist or automate design tasks which may be difficult to solve algorithmically. Several efforts have been made to develop such advanced CAD systems for various design areas using expert system techniques [1][2][3][4].

The authors have developed a CAD system in which register transfer level design is described in DDL, and verified with a simulator and a verifier [5][6]. The DDL description is automatically translated into tables representing conditional operations of each circuit component. However, gate level design according to the tables is done manually. This system was experimentally used to design an adapter between the FACOM Alpha (a LISP machine [7]) and a FACOM M or S series host computer. The experiment revealed that manually optimizing the gate-level design introduces many errors. In addition, designers, who has done gate-level design manually from function diagrams for electronic telephone switching system, had need of a tool to automate synthesis tasks.

The above background motivated us to develop better tools for gate-level design [8][9][10]. The DDL translator generates a technology-independent functional design from a DDL description. A logic circuit synthesizer, DDL/SX, transforms the function diagram into a conventional technology-dependent gate-level design (Figure 1).

The process of synthesizing a logic circuit from a functional design is not clear enough to establish a fixed algorithm. Design constraints and optimization techniques depend greatly on the target technology. Therefore we implemented DDL/SX as a rule-based system. DDL/SX was first developed for automating gate-level design using TTL SSI/MSIs. A CMOS-gate-array version of DDL/SX was developed by replacing the knowledge base and IC library of the TTL SSI/MSI version.

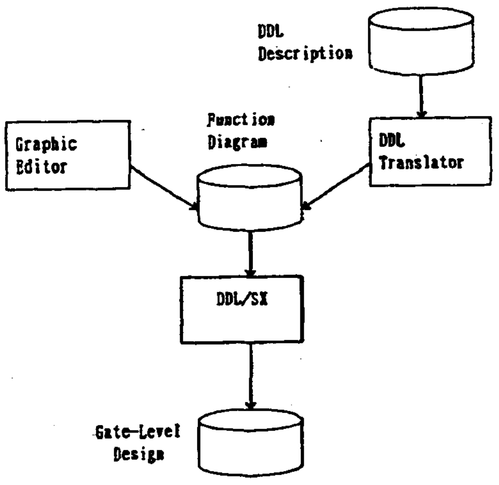


Figure 1. Logic Design System

DEF018277

This paper addresses the CMOS-gate-array version of DDL/SX. Section 2 overviews the system. Section 3 describes rule-based synthesis and the knowledge base. The remaining sections gives examples of synthesis and some experimental results.

## 2. SYSTEM OVERVIEW

The configuration of DDL/SX is shown in Figure 2. A graphic editor [11] is used to define a functional diagram as a collection of interconnected components. A component is an instance of a macro or a circuit. A macro is a technology-independent circuit primitive: a cell with a function specified by a macro may not be available in a cell library. When using these macros, the user need not take into account what kind of cells are available in the cell library. Using a component as an instance of a circuit, a designer can define a circuit in a hierarchical manner.

DDL/SX selects an appropriate macro expansion method by taking account of the physical requirement and available cells (a unit cell consists of several basic cells ; a basic cell is a 2-input NAND gate. Unit cells will be called cells.) in a cell library, and then replaces a macro with a configuration of cells to implement the macro's function. After the macro expansion, the system optimizes the design by removing redundant cells, and detects and eliminates violations of design constraints such as fanout. These tasks are performed by applying rules from the knowledge-base. The knowledge-base is modular and consists of knowledge sources (KSs). Each knowledge source contains a set of rules for a specific task. In the next step, DDL/SX places components and routes lines between them on design sheets automatically, to print out the synthesized circuit. Final results are input to a conventional physical design CAD system for CMOS gate arrays.

In the debug mode, invoked knowledge sources and applied rules can be displayed on a screen. Furthermore, break points can be set for checking the intermediate synthesized results and modifying the knowledge-base. In addition to these debugging facilities provided by ESHELL (Fujitsu's Expert SHELL), a tool for building expert systems, we developed a program which automatically displays a sub-circuit under synthesis. This program shows the diagrams on a graphic display terminal before and after the application of a rule, and makes it easy to understand the effect of the applied rule.

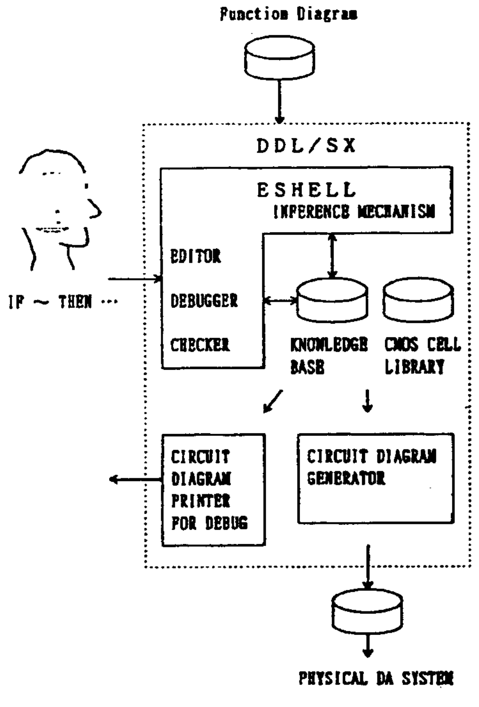


Figure 2. DDL/SX

## 3. Rule-based Synthesis

### 3.1 Production System

ESHELL is a general-purpose tool for building expert systems. It provides the kernel of a production system based on a "blackboard model" [12][13], and an environment which facilitates knowledge base construction. The knowledge base is modularized, consisting of multiple knowledge sources with a set of related rules and information for controlling application of the rules. The inference mechanism searches a knowledge source whose precondition matches the focused event. The right hand side (action part) of a rule in the knowledge source are executed if the left hand side (condition part) of the rule is satisfied. The actions add information or modify it on a blackboard and issue events to invoke other KSs. In this way, KSs cooperate each other to solve the problem.

DDL/SX first focuses on a component in a functional diagram, and copies its function and connections onto the blackboard as a problem (Figure 3). Each rule either solves the problem or decomposes it into subproblems. There is a knowledge source which checks whether all subproblems have been solved. If so, the functional diagram is modified according to the solutions. Otherwise, the KS activates the appropriate KS for synthesis, to solve any remaining subproblems.

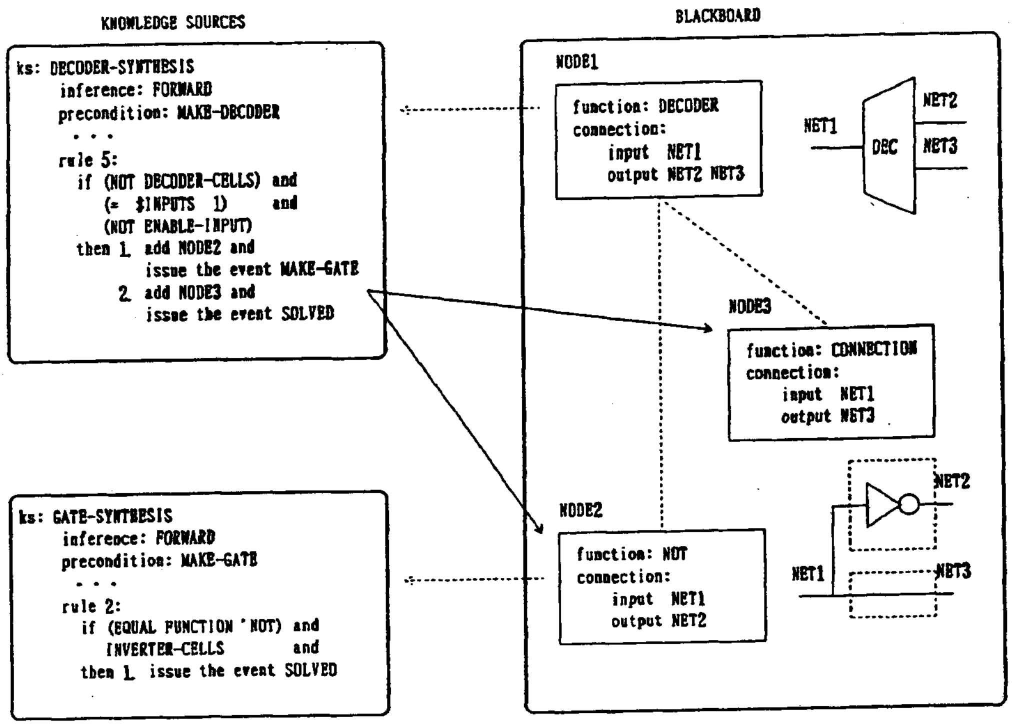


Figure 3. Synthesis Method

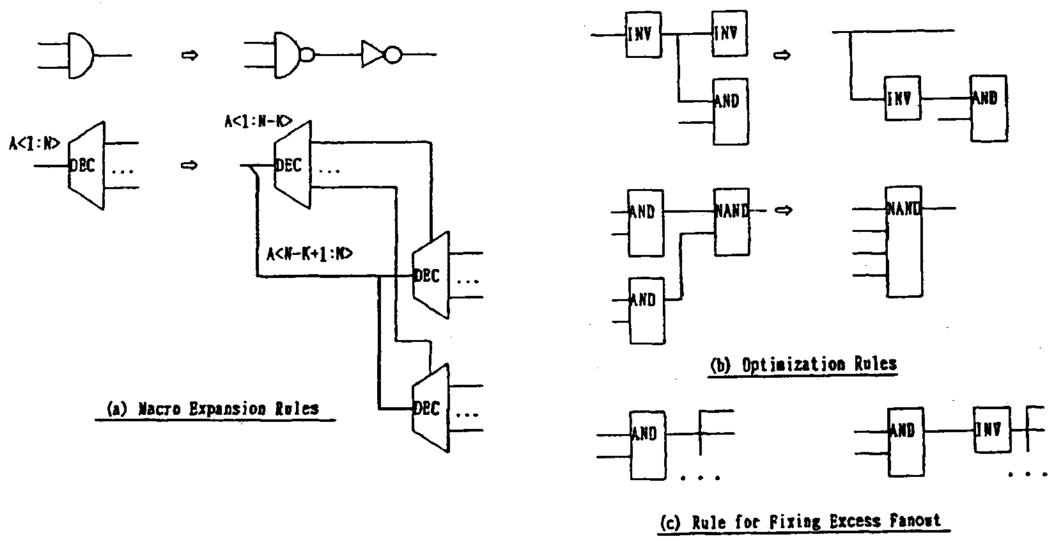


Figure 4. Examples of rules

## 3.2 SYNTHESIS KNOWLEDGE

Synthesis knowledge is acquired through interviews with expert designers. This knowledge can be classified into the following categories:

1) Macro expansion

Macro expansion rules are knowledge about how to organize cells in order to implement a function of a macro. Rules select a method to construct a subcircuit taking into account the macro's function, physical requirement such as space, and the functions available in a specific CMOS gate array family. Some examples follow (Figure 4(a) ):

* If no AND cells match the relevant AND-macro, and there are NAND cells with the same number of input pins
  then make a NAND and an inverter and connect them.

* If no decoder cells match the relevant decoder macro, there are k-bit decoder cells with an enable input, and k is less than n
  then make $2^{n-k}$ k-bit decoders and make a (n - k)-bit decoder which controls the above decoders.

2) Optimization

Optimization rules are for removing redundant cells and for replacing a group of cells with a single cell (Figure 4(b)). Examples are:

* If the relevant object is an inverter and the number of inverters connected to its output is greater than that of non-inverter components
  then remove all inverters, including the relevant inverter and insert inverters at the inputs of all non-inverter components.

* If the relevant component is a NAND cell and another NAND cell is available with the right number of inputs to replace the AND cells at the component's inputs
  then replace both the relevant NAND cell and the AND cells with the new NAND cell.

3) Constraint check

Constraint check rules are for detecting and eliminating violations of design constraints such as fanout. An example follows (Figure 4(c)):

* If the relevant object cannot drive all the gates connected to its output, there is no cell of the same kind that can drive them, and buffer cells are available
  then insert a buffer cell at the output of the relevant object.

4) Miscellaneous rules

To interface the LSI with external circuits, I/O buffer cells and clock buffers must be inserted. Unused pins of components should be connected to dummy cells which represent connections to ground or pull-up circuits. Scan path design rules are also included.

5) Scheduling synthesis tasks

Rules for scheduling the synthesis tasks and checking whether the problem has been solved are in this category.

## 3.3 IMPLEMENTATION

Circuit under synthesis and knowledge about CMOS cells are represented by frames [11] because this knowledge is static. Synthesis tasks, such as hierarchical expansion, which have clear and fixed procedures are written in UtiLisp (University of Tokyo Interactive LISP). Rules described in the previous section are coded in the ESHELL's syntax by using the LISP programs.

Macros of the same type can be expressed by one component in the function diagram. DDL/SX does not partition the component bit-wise until the logic diagram generation phase. So a rule is applied to multiple functions represented by such a component at a time and this improves the processing speed of DDL/SX.

The configuration of the implemented knowledge sources is shown in Figure 5. Arrows indicate which knowledge sources can be invoked from a given knowledge source. For example, the knowledge source INITIAL-TASK is invoked first. It records the circuit name and invokes KS SYNTHESIS.

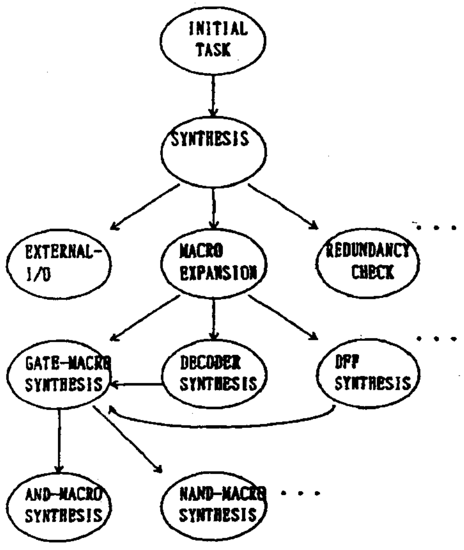


Figure 5. Invocation of knowledge sources

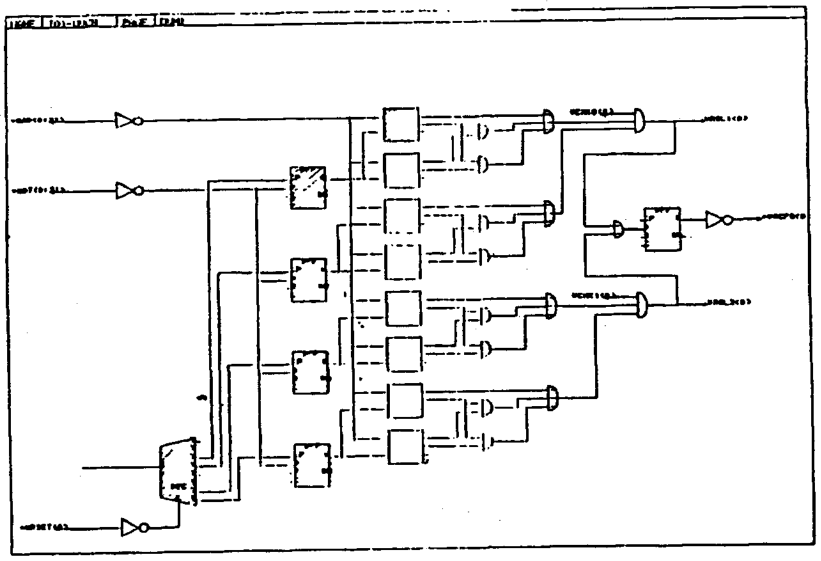


Figure 6. Function Diagram

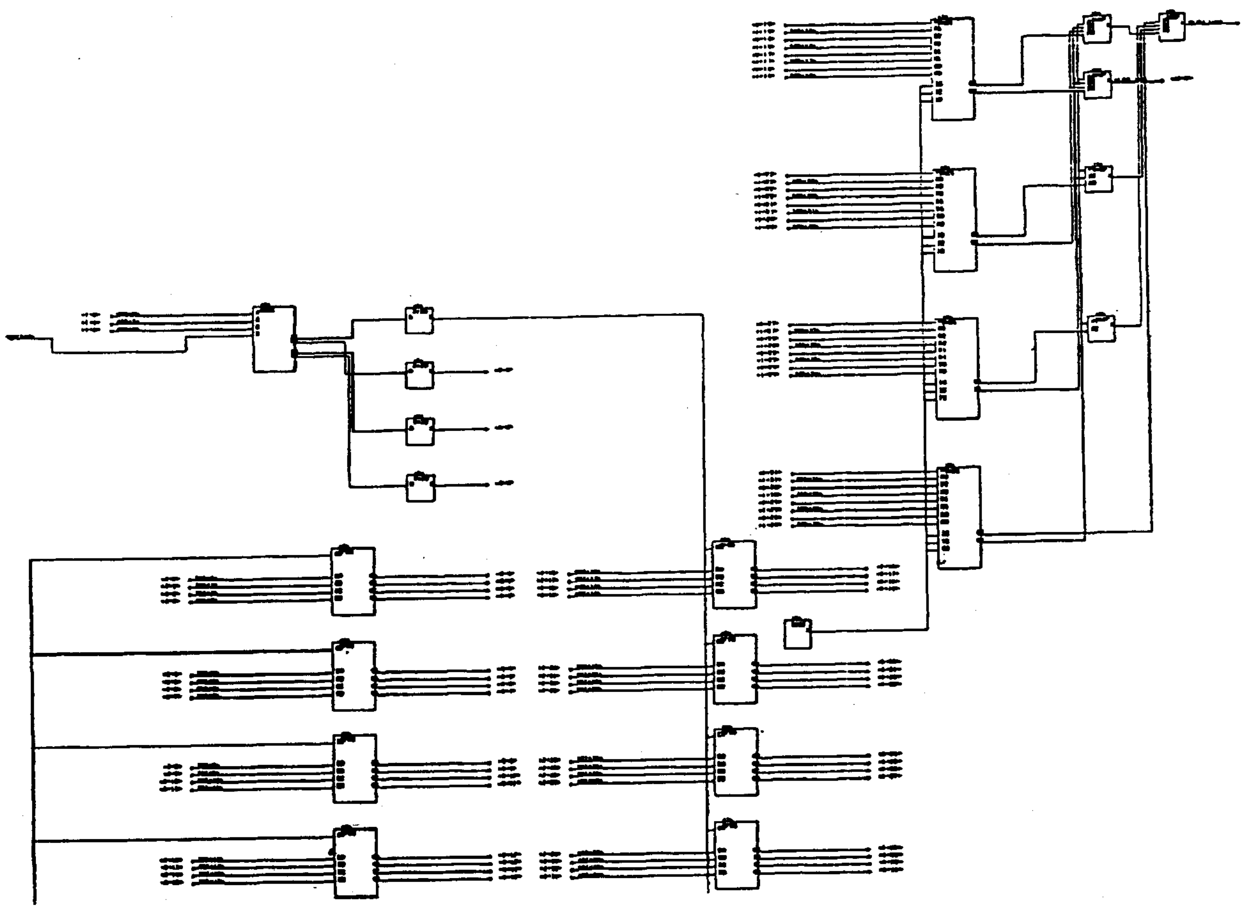


Figure 7. Generated Gate-Level Design

Then the KS SYNTHESIS looks for macros to be expanded and ,if there are any, invokes KS MACRO-EXPANSION. If there is no such macro, KS REDUNDANCY-CHECK is invoked to proceed with optimization. KS MACRO-EXPANSION invokes knowledge sources such as GATE-SYNTHESIS, DFF-SYNTHESIS, and DEC-SYNTHESIS according to the function of the macro.

KSs are applied until there are no macros to be expanded in the functional diagrams. Then, KSs, such as optimization, are invoked in the same way, and finally logic diagrams are generated. DDL/SX has been implemented using UtiLisp under OS IV/F4 MSP on FUJITSU's M series computers. It also runs on the FACOM ALPHA. At present, there are 130 rules in 32 knowledge sources.

## 4. SYNTHESIS EXAMPLE

An example of a functional diagram is shown in Figure 6. It describes a part of an electronic telephone switching system. DDL/SX generated 15 pages of logic diagrams from the input. The logic diagram corresponding to the shaded portions in Figure 6 is shown in Figure 7. The system recognized that two 4-bit DFF cells were available and selected one of them for making a 32-bit DFF with minimum space. So the DFF macros with 32 bits data in Figure 6 are replaced with eight 4-bit DFF cells in Figure 7. The 3-bit decoder macro is changed to the configuration of a 3-bit decoder cell and inverters. These inverters are inserted to adjust the polarity of output signals from the cell to the functional specification.

## 5. EXPERIMENTAL RESULTS

DDL/SX has been used experimentally to evaluate its performance.

The number of macros in functional diagrams and the number of generated unit cells in the corresponding logic diagrams are shown in Table 1. Generally, the amount of description in a functional diagram is about 20 to 60% that in the corresponding conventional logic diagram (Table 1). This makes it easier to review designs and understand their functions.

Table 2 compares automatic and manual gate level designs. The number of basic cells is almost the same for both automatic and manual design.

The system runs on a FACOM M-380 (15 MIPS). The CPU time for synthesis is listed in Table 3. It took eight hours to synthesize a 2000-gate circuit, including time for input and modification of the functional diagram. Manual design required about 30 hours. Therefore it is expected that logic design time from functional diagrams can be reduced to one-fourth of the conventional design time. The average CPU times per rule are also listed in Table 2.

The garbage collection method used in UtiLisp limits the available memory space to half so that DDL/SX can synthesize up to 8000-gate circuits. Extended frame access LISP functions, which use a disc and is similar to a virtual memory system, can be used for synthesizing more large circuits.

Table 1. Function and Logic Diagrams

| Circuit | No. of macros in function dia. | No. of unit cells in logic dia. | Circuit type |
|---|---|---|---|
| A | 39 | 179 | data path |
| B | 45 | 208 | data path |
| C | 68 | 140 | control |
| D | 224 | 371 | control |

Table 2. Automatic and Manual Design

| Circuit | No. of types of unit cell | No. of unit cells | No. of basic cells |
|---|---|---|---|
| A (man) | 7 | 179 | 2045 |
| (auto) | 7 | 192 | 2058 |
| B (man) | 10 | 208 | 2225 |
| (auto) | 13 | 190 | 2228 |
| C (man) | 10 | 140 | 682 |
| (auto) | 12 | 139 | 694 |
| D (man) | 23 | 371 | 2153 |
| (auto) | 31 | 303 | 2134 |

Table 3. Run time

| Circuit | CPU time (S) | No. of applied rules | CPU time/ rule (S) |
|---|---|---|---|
| A | 2.4 | 420 | 0.006 |
| B | 4.9 | 478 | 0.010 |
| C | 2.9 | 527 | 0.006 |
| D | 15.0 | 1756 | 0.009 |

## 6. CONCLUSION

A rule-based logic synthesis system has been presented to eliminate error-prone and time consuming tasks in conventional logic circuit design. The system generates logic diagrams for CMOS gate arrays from functional specifications. Expert system techniques make it easy to add new design knowledge and to adapt the system to different technologies. For example, it took only one and a half man-years to develop the CMOS version, compared to four man-years for the original TTL IC version.

The circuits synthesized by the system are almost as good as those produced by human designers. Synthesis of circuits with about 2000 basic cells takes 2 to 15 seconds of CPU time. This system should greatly improve reliability and productivity.

## ACKNOWLEDGEMENTS

The authors would like to thank Mr. K. Yuasa and Mr. T. Kakuta of FUJITSU LIMITED for their corporation. They would also like to thank Mr. S. Sato and Dr. T. Uehara for their encouragement.

## REFERENCES

[1] H. Brown, et al., "Palladio: A Explanatory Environment for Circuit Design," IEEE Computer, Vol. 16, No. 12 1983.

[2] L. I. Steinberg et al., "The Redesign System: A Knowledge-Based Approach to VLSI CAD," IEEE Design & Test, pp.45-54, Feb. 1985.

[3] T. J. Kowalski et al., "The VLSI Design Automation Assistant: What's in a Knowledge Base," Proc. 22th DA Conf. pp.252-258, 1985.

[4] A. J. de Geus et al., "A Rule Based System for Optimizing Combinational Logic," IEEE Design & Test, pp. 22-32, Aug. 1985.

[5] N. Kawato et al., "Design and Verification of Large Scale Computers by Using DDL," Proc. 16th DA Conf., pp. 360-366, 1979.

[6] F. Maruyama et al., "Hardware Verification and Design Error Diagnosis," FTCS-10, pp. 59-64, 1980.

[7] H. Hayashi et al., "ALPHA: A High Performance LISP Machine Equipped with A New Stack Structure and Garbage Collection System," Proc. 10th Int. Symp. on Computer Architecture, 1983.

[8] N. Kawato et al., "An Interactive Logic Synthesis System Based Upon AI Techniques," Proc. 19th DA Conf. pp. 858-864, 1982.

[9] N. Kawato et al., "DDL/SX: A Rule-Based Expert System for Logic Circuit Synthesis," ISCAS 1985.

[10] T. Uehara, "A Knowledge-based Logic Design System," IEEE DESIGN & TEST, pp. 27-34, Oct. 1985.

[11] T. Saito et al.,"A CAD System for Logic Design Based on Frames and Demons," Proc. 18th DA Conf. , pp. 451-456, 1981.

[12] Frederic Hayes-Roth, Donald A. Waterman, Douglas B. Lenat, "Building Expert Systems," Addison-Wesley Publishing Company.

[13] H. P. Nii et al., "AGE (Attempt to Generalize): A Knowledge-Based Program for Building Knowledge-Based Programs," IJCAI 6, pp.645-655, 1979.

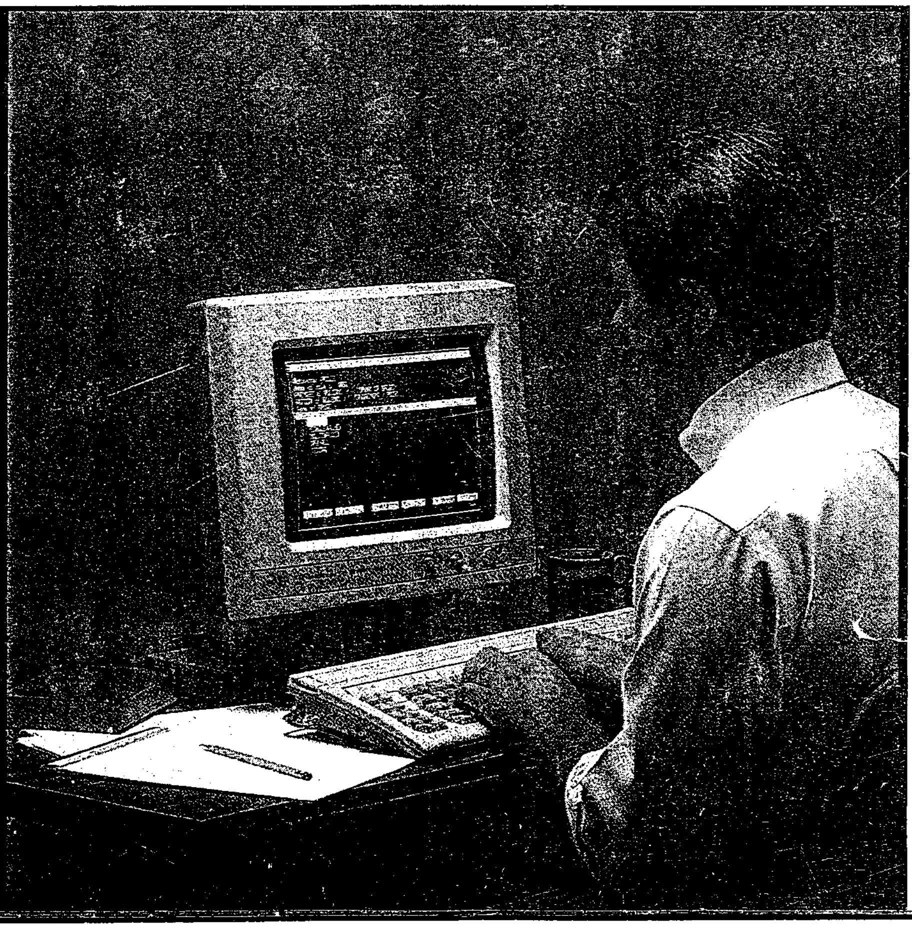
30

2738

# EXPERT SYSTEMS

A NON-PROGRAMMER'S GUIDE TO DEVELOPMENT AND APPLICATIONS

PAUL SIEGEL



FIRST EDITION

FIRST PRINTING

Library of Congress Cataloging in Publication Data

Siegel, Paul.
Expert systems.

Bibliography: p.
Includes index.
1. Expert systems (Computer science)
I. Title.
QA76.76.E95S54 1986 006.3'3 86-6020
ISBN 0-8306-2738-3

is the one for mammal. The top relationship is that *mammal* is *a-kind-of animal*; this relationship acts as a pointer to the animal frame. The skin-cover and activity relations are also shown in the mammal frame. The carnivore and bird frames are related to the semantic network in a similar way.

Two big advantages of frames over semantic networks are that frames may be used for partitioning a complex domain and for storing procedures in addition to descriptive data.

## Rules

One of the simplest ways to present knowledge is by rules. Thus, the primary chunk of knowledge given by the mammal frame is:

If animal has hair,
and animal produces milk,
then animal is a mammal.

The rule is presently the most popular method of knowledge representation and is the one I concentrate on for the remainder of this book. The rule is popular because it is:

☐ Simple.
☐ Modular.
☐ Of appropriate size.
☐ Procedural as well as descriptive.

**Simple.** It is easy to express, to understand and to work with.

**Modular.** Each rule expresses a separate thought and it may be changed or modified without affecting other rules.

**Of Appropriate Size.** Relations in semantic networks seem to be too detailed. Frames seem to be too broad. Rules are or could be made the correct size. For instance, the data within the bird frame could be part of, not one, but two rules:

1—If animal has feathers,
then animal is bird.
2—If animal flies
and animal lays eggs,
then animal is bird.

**Procedural as Well as Descriptive.** The rules presented up to now are descriptive. However, rules may refer to procedure as well. This will become obvious as we go along.

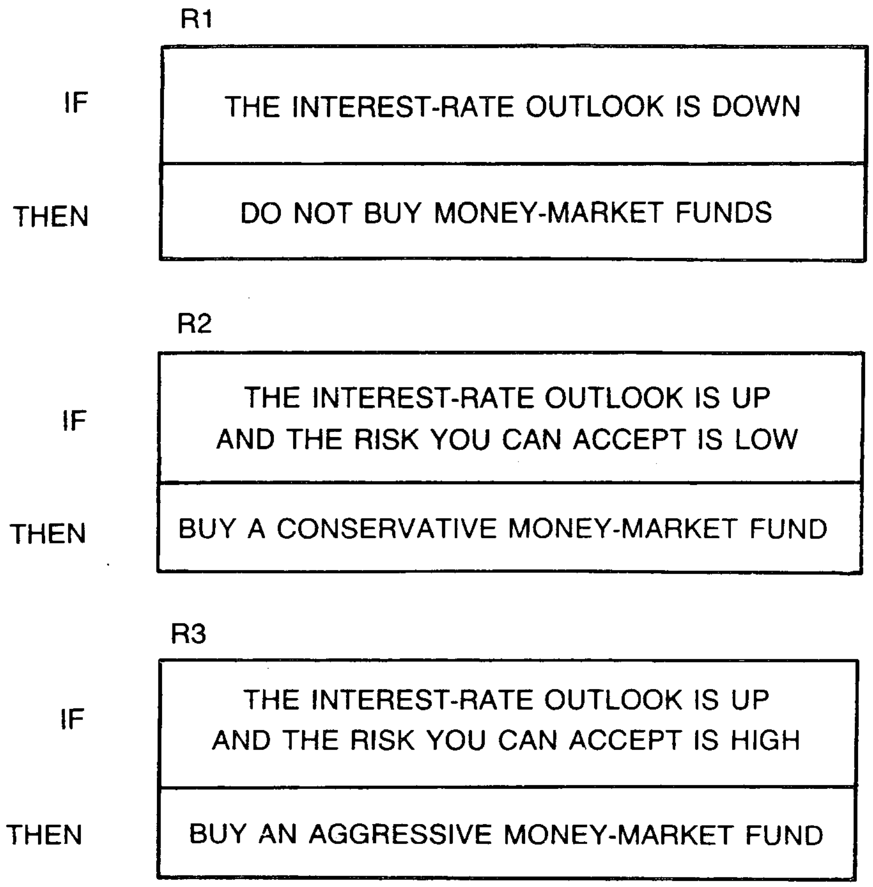

R1
IF
THE INTEREST-RATE OUTLOOK IS DOWN
THEN
DO NOT BUY MONEY-MARKET FUNDS
R2
IF
THE INTEREST-RATE OUTLOOK IS UP AND THE RISK YOU CAN ACCEPT IS LOW
THEN
BUY A CONSERVATIVE MONEY-MARKET FUND
R3
IF
THE INTEREST-RATE OUTLOOK IS UP AND THE RISK YOU CAN ACCEPT IS HIGH
THEN
BUY AN AGGRESSIVE MONEY-MARKET FUND

Fig. 1-5. IRA-investment rules.

## RULES-OF-THUMB

We often speak of *rules-of-thumb*, meaning guidelines we follow in our reasoning. "An apple a day keeps the doctor away" and "a stitch in time saves nine" are rules-of-thumb.

Rules-of-thumb are easily converted into rules more formally expressed, something which must be done to make the rules manipulatable by computer. Figure 1-5 presents three such rules which are part of an IRA-investment expert system.

A rule consists of two parts: an IF-part and a THEN-part. A rule states a simple relationship: IF certain conditions are true THEN a given conclusion follows. The first rule, R1, states that IF the clause "the interest-rate outlook is down" is true, THEN the conclusion "do not buy money-market funds" follows. Rule R2 is similar except that it has two clauses in the IF-part. It states that IF both clauses, "the interest-rate outlook is up" *and* "the risk you can accept is low" are true, THEN the conclusion "buy a conservative money-market fund," follows. Rule R3 is similar to R2, except that it indicates what should be done if the interest-rate outlook is up and the risk you can accept is high; in this case, buy an aggressive money-market fund.

Figure 1-6 presents two more rules: R4 and R5. These rules are similar in structure to R1, R2, and R3. Note, however, that the THEN-part of R4 is the same as the second IF-clause of R2, and the THEN-part of R5 is the same as the second IF-clause of R3.

In general, the IF-part may have any number of condition clauses. Similarly, the THEN-part may consist of any number of conclusion clauses. In practice, try to limit the size of both the IF-part and the THEN-part to keep the logic simple.

Rules are modular, pithy chunks of knowledge, which can be replaced or modified without affecting other rules. Rules are the basic building blocks of the *knowledge base* which stores your expertise. Expert systems built around such knowledge bases are flexible, adaptable to changing conditions and able to handle complex problems.

## REASONING

Machine reasoning is the path the computer follows as it traces rules through a knowledge base. When the machine sequences forward from facts to final conclusions, or goals, the process is called *forward reasoning* (or forward chaining). When the machine sequences backward from final conclusions, or goals, to facts, the process is called *backward reasoning* (or backward chaining).

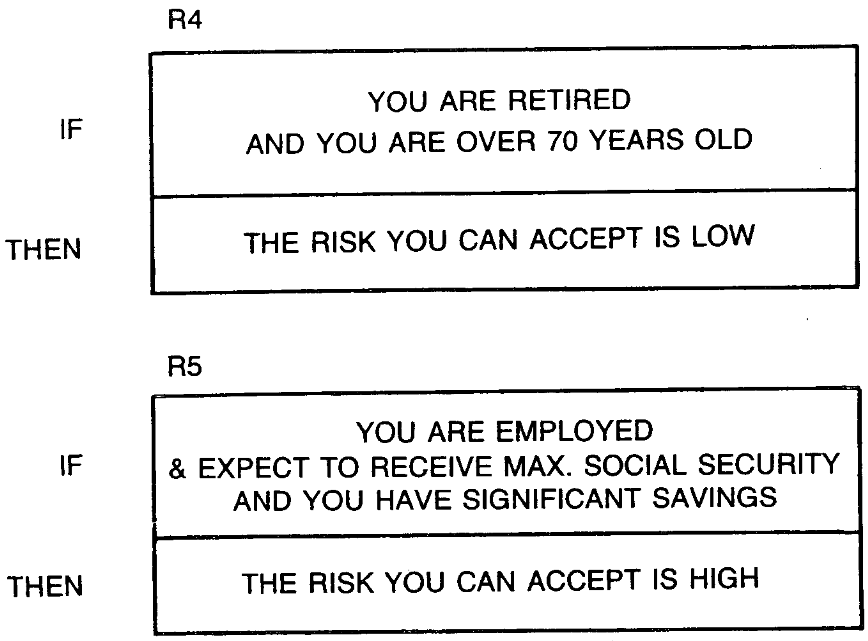

R4
IF
YOU ARE RETIRED
AND YOU ARE OVER 70 YEARS OLD
THEN
THE RISK YOU CAN ACCEPT IS LOW
R5
IF
YOU ARE EMPLOYED
& EXPECT TO RECEIVE MAX. SOCIAL SECURITY
AND YOU HAVE SIGNIFICANT SAVINGS
THEN
THE RISK YOU CAN ACCEPT IS HIGH

Fig. 1-6. Two more IRA-investment rules.

clusion, it answers with the rule it used. In the previous example, if the client asks how it determined that the client should "Buy aggressive M.M.", it answers by displaying rule R3.

## MAJOR CHARACTERISTICS OF EXPERT SYSTEMS

The following are the major characteristics of a rule-based expert system which you can develop with the aid of an Expert-System Builder such as the Personal Consultant without doing any programming:

### Knowledge Base

Your knowledge base consists of a collection of rules plus other data representing your knowledge in a specific domain. These rules consist of IF-parts stating the conditions and THEN-parts giving the conclusions.

### Separate Reasoning Capability

The reasoning capability is separate from the knowledge base. The reasoning capability, which for the Personal Consultant is primarily backward reasoning, comes built into the Expert-System Builder. But it is copied to a diskette to be made part of the expert system.

### System-Client Communication

One way the expert system may activate appropriate rules is by initiating a dialogue with the client. It asks a sequence of questions. Each question is guided by the client's answers to previous questions as well as by the rules and associated information guidelines you, as the developer, have previously stored in the system. Then the expert system communicates advice to the client. The expert system may also communicate directly with other software packages.

### Uncertainty

The expert system can handle uncertainty in facts and in rules relating these facts by the use of certainty factors or other functions.

### Explanation

An expert system can explain the reasoning it intends to follow or has followed to reach a conclusion The first is often given in answer to a why, the second in answer to a how.

31

# EXPERT SYSTEMS

# TOOLS & APPLICATIONS

Paul Harmon Rex Maus
William Morrissey

The figures appearing in this book, unless otherwise noted, originally appeared in the *Expert Systems Strategies* newsletter and are reproduced with the permission of Harmon Associates, San Francisco, CA.

Figures 3.1, 4.1, and 10.2, and Tables 2.5 and 3.1 are modifications of figures and tables that originally appeared in *Expert Systems: Artificial Intelligence in Business*, by Paul Harmon and David King (John Wiley & Sons, 1985).

Publisher: Stephen Kippur
Editor: Therese A. Zak
Managing Editor: Ruth Greif
Editing, Design & Production: G&H SOHO, Ltd.

**Library of Congress Cataloging-in-Publication Data**

Harmon, Paul.
Expert systems tools and applications /
Paul Harmon, Rex Maus, William Morrissey.
p. cm.
1. Business—Data processing. 2. Expert systems (Computer science) I. Maus, Rex. II. Morrissey, William. III. Title.
HF5548.2.H367 1988
650'.028'5633—dc19 87-17608
CIP

ISBN 0-471-83951-5
ISBN 0-471-83950-7 (pbk.)

Printed in the United States of America

88 89 10 9 8 7 6 5 4 3 2 1

DEF079992

haustive search can become impossible. Methods are domain independent strategies like "generate and test." Strong methods exploit domain knowledge to achieve greater performance. This is usually accomplished by avoiding exhaustive search in favor of exploring a few likely solutions.

**Problem Space.** A conceptual or formal area defined by all of the possible states that could occur as a result of interactions between the elements and operators that are considered when a particular problem is being studied.

**Procedural versus Declarative.** Procedures tell a system what to do (e.g., Multiply A times B and then add C). Declarations tell a system what to know (e.g., V=IR).

**Programming Environment** (Environment). A programming environment is about halfway between a language and a tool. A language allows the user complete flexibility. A tool constrains the user in many ways. A programming environment, like OPS5, provides a number of established routines that can facilitate the quick development of rule-based programs.

**PROLOG.** A symbolic or AI programming language based on Predicate Calculus. PROLOG is the most popular language for AI research outside North America.

**Prototype.** In expert systems development, a prototype is an initial version of an expert system that is developed to test effectiveness of the overall knowledge representation and inference strategies being employed to solve a particular problem.

**Pruning.** In expert systems, this refers to the process whereby one or more branches of a decision tree are "cut off" or ignored. In effect, when an expert systems consultation is under way, heuristic rules reduce the search space by determining that certain branches (or subsets of rules) can be ignored.

**Reasoning.** The process of drawing inferences or conclusions.

**Representation.** The way in which a system stores knowledge about a domain. Knowledge consists of facts and the relationships between facts. Facts, rules, objects, and networks are all formats for representing knowledge.

**Robotics.** The branch of AI research that is concerned with enabling computers to "see" and "manipulate" objects in their surrounding environment. AI is not concerned with robotics as such, but it is concerned with developing the techniques necessary to develop robots that can use heuristics to function in a highly flexible manner while interacting with a constantly changing environment.

**Rule** (If-Then Rule). A conditional statement of two parts. The first part, composed of one or more if clauses, establishes conditions that must apply if a second part, composed of one or more than clauses, is to be acted upon. The clauses of rules are usually A-V pairs or O-A-V triplets.

**Rule-Based Program.** A computer program that represents knowledge by means of rules.

**Runtime Version or System.** Knowledge system building tools allow the user to create and run various knowledge bases. Using a single tool, a user might create a dozen knowledge bases. Depending on the problem the user was facing, he or she would load an appropriate knowledge base and undertake a consultation. With such a tool the user can easily modify a knowledge base. Some companies will want to develop a specific knowledge base and then produce copies of the tool and that specific knowledge base. Under these circumstances the organization will not want the user to have to "load" the knowledge base, nor will they want the user to be able to modify the knowledge base. When an expert system building tool is modified to incorporate a specific knowledge base and to deactivate certain programming features, the resulting system is called a runtime system or a runtime version.

**Search** and **Search Space.** See Problem-Solving and Problem Space.

**Semantic.** Refers to the meaning of an expression. It is often contrasted with syntactic, which refers to the formal pattern of the expression. Computers are good at establishing that the correct syntax is being

DEF080261

32

# Expert Systems

## Principles and case studies

edited by

RICHARD FORSYTH

Knowledge
+ Inference
= System

CHAPMAN AND HALL COMPUTING

# EXPERT SYSTEMS
## *Principles and case studies*

EDITED BY

Richard Forsyth
*Polytechnic of North London*

LONDON NEW YORK
Chapman and Hall

*First published 1984 by*
*Chapman and Hall, Ltd*
*11 New Fetter Lane, London EC4P 4EE*
*Published in the USA by*
*Chapman and Hall*
*733 Third Avenue, New York NY 10017*

© 1984 Chapman and Hall
© 1984 Tom Stonier (Chapter 13)

*Printed in Great Britain at the University Press, Cambridge*

ISBN 0 412 26270 3 (hardback)
ISBN 0 412 26280 0 (paperback)

**British Library Cataloguing in Publication Data**

Expert systems.
1. Expert systems (Computer science)
I. Forsyth, Richard
001.64 QA76.9.E96

ISBN 0-412-26270-3
ISBN 0-412-26280-0 (pbk.)

**Library of Congress Cataloguing in Publication Data**

Expert systems.

Bibliography: p.
Includes index.
1. Expert systems (Computer science) I. Forsyth, Richard.
QA76.9.E96E96 1984 001.64 84-15529
ISBN 0-412-26270-3
ISBN 0-412-26280-0 (pbk.)

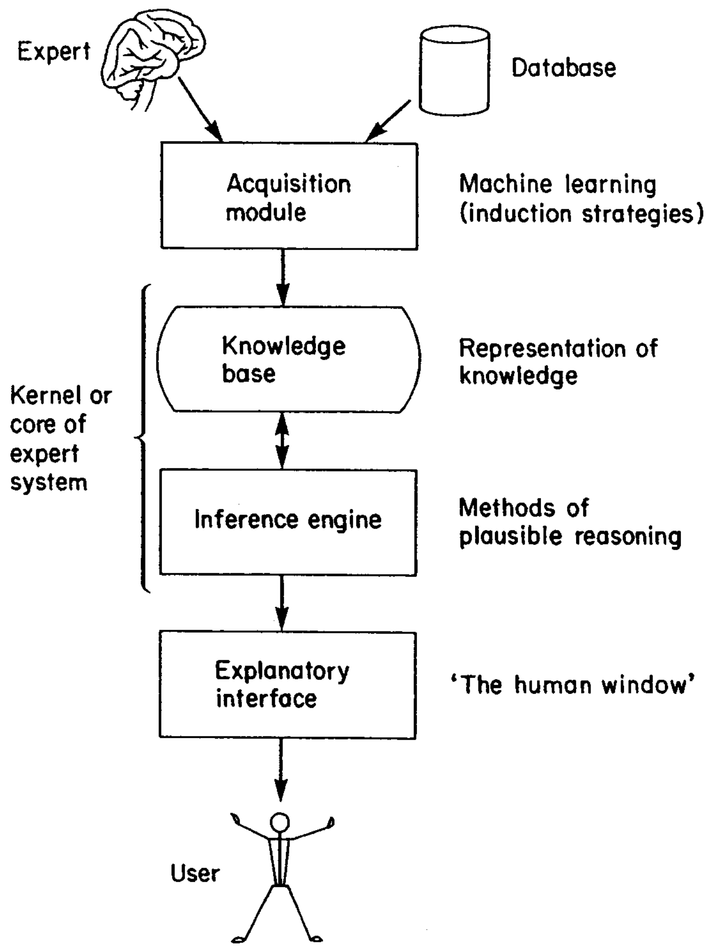

Expert
Database
Acquisition module
Machine learning (induction strategies)
Knowledge base
Representation of knowledge
Kernel or core of expert system
Inference engine
Methods of plausible reasoning
Explanatory interface
'The human window'
User

*Fig. 2.1* A typical expert system

## 2.3 THE KNOWLEDGE BASE

A knowledge base contains facts (or assertions) and rules. Facts are short-term information that can change rapidly, e.g. during the course of a consultation. Rules are the longer-term information about how to generate new facts or hypotheses from what is presently known.

How does this approach differ from conventional database methodology? The major difference is that a knowledge base is more creative. Facts in a database are normally passive: they are either there or not there. A knowledge base, on the other hand, actively tries to fill in the missing information.

Production rules are a favourite means of encapsulating rule-of-thumb knowledge. These have a familiar IF–THEN format, for example:

*Rule 99*
IF the home team lost their last home game, AND the away team won their last home game
THEN the likelihood of a draw is multiplied by 1.075; the likelihood of an away win is multiplied by 0.96.

But remember, these rules are not embedded in program code: they are data for a high-level interpreter, namely the inference engine.

Production rules are not the only way to represent knowledge. Other systems have used decision trees (e.g. ACLS), semantic nets (e.g. PROSPECTOR) and predicate calculus. Since one form of predicate calculus – 'Horn clauses' – is built into PROLOG together with a free theorem prover, this latter representation shows signs of gaining in popularity. At some very deep level all kinds of knowledge representation must be equivalent, but they are not all equally convenient. When in doubt, the best plan is to choose the simplest you can get away with.

## 2.4 THE INFERENCE ENGINE

There is some controversy in the field between supporters of 'forward chaining' versus 'backward chaining' as overall inference strategies. Broadly speaking, forward chaining involves reasoning from data to hypotheses, while backward chaining attempts to find data to prove, or disprove, a hypothesis. Pure forward chaining leads to unfocused questioning in a dialog-mode system, whereas pure backward chaining tends to be rather relentless in its goal-directed questioning.

Therefore most successful systems use a mixture of both, and Chris Naylor (1983) has recently described a method known as the Rule Value approach which combines some of the merits of both strategies. I call it 'sideways chaining'. (See also Chapter 6 of this volume.)

Whether your inferencing procedure works primarily backwards or forwards, it will have to deal with uncertain data and this is where things start to get interesting. For too long computer specialists have tried to force the soft edges of the world we actually inhabit (and understand well enough) into the rigid confines of hard-edged computer storage. It has never been a comfortable fit. Now we have means of dealing with uncertainty, in other words with the real world rather than some idealized abstraction that our data-system forces us to believe in.

Indeed, we have too many ways of dealing with uncertainty! There is fuzzy logic, Bayesian logic, multi-valued logic and certainty factors, to name only four. All sorts of schemes have been tried, and the odd thing is that most of them seem to work.

My explanation for this state of affairs is that the organization of